Projet Codepo

Création d'une base de données chargée du suivi médical de patientes atteintes du cancer du col de l'utérus en Bolivie

Année académique 2014-2015

Bayot Madli – Trigaux Anaïs – Verdès Guillem

Table des matières

Remerciements

Nous tenons tout d'abord à remercier la Cellule de coopération de l'Ecole polytechnique de Bruxelles (ou CoDePo) de nous avoir sélectionné et fait confiance pour mener à bien ce projet des plus enrichissants.

Nous souhaiterions aussi remettre nos remerciements au Professeur Antoine Nonclercq et à Loïc Vaes pour le suivi attentif, l'encouragement et les bons conseils prodigués tout au long de la réalisation du système de gestion de base de données en Belgique.

En outre, merci à Denis Steckelmacher, étudiant en première année de master en sciences informatiques, pour l'aide précieuse qu'il nous a fournie du point de vue technique.

Le Professeur Véronique Fontaine de la Faculté de Pharmacie de l'ULB nous a également été d'une grande aide, de par sa connaissance de la réalité de terrain en Bolivie. Au départ, elle a construit un réel pont entre la Belgique et la Bolivie. Petit à petit, à force d'échanges, nous avons pu créer un contact direct avec les locaux.

Nous remercions, par ailleurs, chaleureusement Patricia Rodriguez, chercheuse au laboratoire virologique HPV-HIV de Cochabamba, pour sa motivation, sa disponibilité et sa volonté de faire aboutir ce projet. Elle a tout mis en œuvre pour centraliser les données et points de vue des différents laboratoires et centres de santé et nous les faire parvenir. Nous sommes également reconnaissants à Pedro Surriabre, autre membre du laboratoire HPV-HIV, pour nous avoir apporté un second point de vue lors de la création de la base de données.

Abstract

[Rapport : 26 155 mots] This report describes the structure and the methodology followed to design a database to provide a reliable communication platform for a grid of laboratories and hospitals in Cochabamba, Bolivia. This database will aid in the reduction of the cervical cancer in this country by facilitating the monitoring of the patients at risk of suffering this disease. The selection of the database management system, the programming language, the interface designer and the support functions required are thoughtfully revised and put in context. An explanation of the structure of the database, its securitization and the application's architecture clarifies its internal functions and the way that will operate to receive, stock and deliver the patient's data. All these tasks are fulfilled taking into account the confidential constraints that the database has to undergo. The outcome is a tailor-made database, as its support functions have been conceived keeping in mind how the patients are being supervised in the South. In addition, it has been constantly redesigned following the futures users feedback.

Keywords : reliable communication platform – Cochabamba – cervical cancer – database management system – interface designer – support functions – confidential constraints.

Ce rapport décrit la structure et la méthodologie suivies pour créer une base de données servant de plateforme de communication fiable et intuitive pour un réseau de laboratoires et hôpitaux dans la région de Cochabamba, Bolivie. Cette base de données apportera sa contribution dans la réduction du taux de mortalité lié au cancer du col de l'utérus dans cette région en facilitant le suivi des patients à risque. Le choix du système de gestion de base de données, du langage de programmation, de la bibliothèque permettant la création des interfaces graphiques et du type de support sont largement développés et mis en contexte dans ce rapport. Une explication de la structure de la base de données, de sa sécurisation et de l'architecture de l'application clarifie ses fonctions internes et sa manière de fonctionner pour recevoir, stocker et envoyer les données relatives aux patientes. Toutes ces tâches sont accomplies en prenant en compte les différentes contraintes de confidentialité que doit subir la base de données. Le résultat est une base de données prête à l'emploi, étant donné que sa conception a été réalisée en prenant en compte la façon dont les patientes sont suivies au Sud et les diverses remarques des futurs utilisateurs.

Mots-clés : platforme de communication fiable – Cochabamba – cancer du col de l'utérus – système de gestion de base de données – interfaces graphiques – support – confidentialité.

1. Introduction

1.1 Mise en situation

La Bolivie est l'un des pays du monde où le cancer du col de l'utérus à une des plus hautes incidences. La proportion de femmes atteintes par ce cancer en Bolivie s'élève à 55 femmes affectées pour 100.000. A titre de comparaison, le taux européen est proche de 10 femmes pour 100.000. Ces chiffres alarmants peuvent notamment s'expliquer par le fait que, pour des raisons culturelles, la population bolivienne a peu recours à des techniques de contraception telles que le préservatif. Cependant, la possibilité d'une influence génétique a également été mise en avant pour justifier un taux de cancer du col de l'utérus aussi important, en comparaison à d'autres pays ayant le même niveau de développement[1].

La situation est d'autant plus tragique que le taux de mortalité lié à la maladie est extrêmement élevé en Bolivie, alors que l'on peut guérir ce cancer avec plus de 99% de succès s'il est dépisté à temps. À l'heure actuelle, les outils de dépistage ont une spécificité et une sensibilité excellentes. En Belgique, le taux de mortalité lié à ce type de cancer tend à être nul, nouvelle très encourageante pour les nouvelles patientes. Il existe donc une corrélation évidente entre la diminution de l'incidence du cancer du col de l'utérus et la couverture du dépistage des lésions précancéreuses et cancéreuses.

Afin d'améliorer le dépistage des lésions du col de l'utérus en Bolivie, la faculté de Pharmacie de l'ULB a entamé une collaboration avec l'Université Mayor San Simón (UMSS) à Cochabamba. Cette collaboration a pour but de développer un laboratoire de dépistage virologique par la détection des HPV-HR (papillomavirus humain à haut-risque), former les infirmières et gynécologues au dépistage visuel et sensibiliser la population à la possibilité d'un dépistage précoce et d'un traitement. Il s'agit en outre d'un projet pilote qui pourrait s'étendre à d'autres régions de la Bolivie s'il s'avère concluant.

Le dépistage peut se faire par un test de frottis Pap, mais trop peu de personnel médical n'est formé pour pratiquer cet examen. De plus, le matériel nécessaire à sa mise en œuvre n'est pas accessible en zone rurale. Heureusement, d'autres dépistages peuvent se faire dans n'importe quel centre de santé simplement par injection d'un produit tel que de l'acide acétique ou de l'iode, qui met visuellement en avant les cellules tumorales. La dernière possibilité mise en place est prometteuse puisqu'elle peut se faire par la patiente elle-même à domicile. Un coton-tige doit être inséré à l'entrée du vagin de la patiente afin de prélever des cellules, puis ce coton sera déposé dans le centre de santé le plus proche avant d'être envoyé vers le laboratoire de tests de la province. Des chercheurs en virologie pourront ensuite extraire l'ADN de ces cellules et chercher s'il y a une trace ou non d'ADN virale. À

[1] POPULATION REFERENCE BUREAU (PRB), ALLIANCE FOR CERVICAL CANCER PREVENTION (ACCP), *Prévenir le cancer du col de l'utérus de par le monde,* 2004.

l'avenir, de plus en plus de techniques seront employées et cumulées pour traiter les patientes en fonction de leur type de lésions.

Si ce projet montre d'ores et déjà des résultats très prometteurs, la transmission de l'information s'avère actuellement difficile et entrave le bon déroulement des opérations. En effet, un grand nombre d'intervenants sont impliqués dans le suivi (dépistage et traitement) des patientes et un échange efficace de données est donc indispensable. Le problème est que la plupart des informations et des dossiers médicaux sont encore en version papier et circulent donc de main à main entre les différents centres. Si certains centres sont partiellement informatisés, il n'existe malgré tout aucune plateforme cohérente entre ces différents départements. Cela entraine généralement une lenteur dans la transmission des résultats et conjointement le non suivi d'environ 80% des personnes qui devraient être traitées, et, par extension, une perte de confiance de la population dans le système des soins de santé.

Étant donné l'inefficacité relative du système en place, les responsables du projet à l'UMSS souhaitent disposer d'une solution partiellement informatisée. Celle-ci devra faciliter l'organisation et la circulation des données des patients entre les différents intervenants, de manière intuitive pour l'utilisateur, tout en respectant les contraintes de coût et de confidentialité des données. En d'autres termes, cela permettra une meilleure gestion des données, un suivi plus personnalisé et efficace du patient, un gain de temps considérable, et une meilleure vision du flux des patients, des actes médicaux réalisés, etc. La base de données servira également aux études de surveillance épidémiologique des lésions précancéreuses et cancéreuses, études menées par le Dr. Allende, gynécologue au HMIGU (Hospital Materno Infantil "German Urquidi"). Enfin, les autres objectifs du projet sont la décentralisation et la décongestion des consultances du Dr. Barriga, responsable du Bureau de consultance gynécologique (HMIGU) qui s'occupe du suivi et traitement de toutes les patientes à haut risque de la région.

1.2 Présentation du site bolivien

Cette section a pour but de mettre en avant les différents acteurs présents en Bolivie et leur rôle dans le dépistage et le traitement d'une patiente, afin de détailler l'environnement dans lequel le projet devra fonctionner ainsi que les différentes personnes qui interagiront avec.

Tout d'abord, une patiente est reçue dans un cabinet médical (centres de santé de niveau 1, 2 et 3 et hôpital maternel-infantile HMIGU qui est lui-même de 3ème niveau) par un gynécologue ou une infirmière. S'il existe des possibilités pour que la patiente soit atteinte du cancer du col de l'utérus, il faut exécuter des tests dont les analyses seront effectuées par les laboratoires de cytologie et HPV-HIV. Une fois les résultats obtenus, la patiente doit retourner voir le gynécologue. Pour les patientes à haut risque, un test supplémentaire est nécessaire. Il est géré par le laboratoire d'anatomopathologie qui en fait l'analyse. Là encore,

une fois les résultats révélés, la patiente doit reprendre rendez-vous avec le gynécologue et débute un suivi régulier. Il est donc évident que les laboratoires centraux de virologie HPV-HIV, cytologie et de pathologie ainsi que les médecins traitants et le personnel soignant doivent partager des informations cruciales, telles que les données de contact des patients et les résultats virologiques, cytologiques, pathologiques et cliniques visuels.

Actuellement, les données sont transmises de main à main sur papier au sein des différents centres de santé, du laboratoire de cytologie et de l'hôpital maternel-infantile (HMIGU) dirigé par le Dr. Barriga. En parallèle, le laboratoire de pathologie possède déjà sa propre base de données informatisée et le laboratoire central de virologie HPV-HIV encode les données des patientes dans un tableau Excel.

Il est à noter que les différents laboratoires ainsi que l'hôpital maternel-infantile se situent tous sur le campus universitaire de l'UMSS.

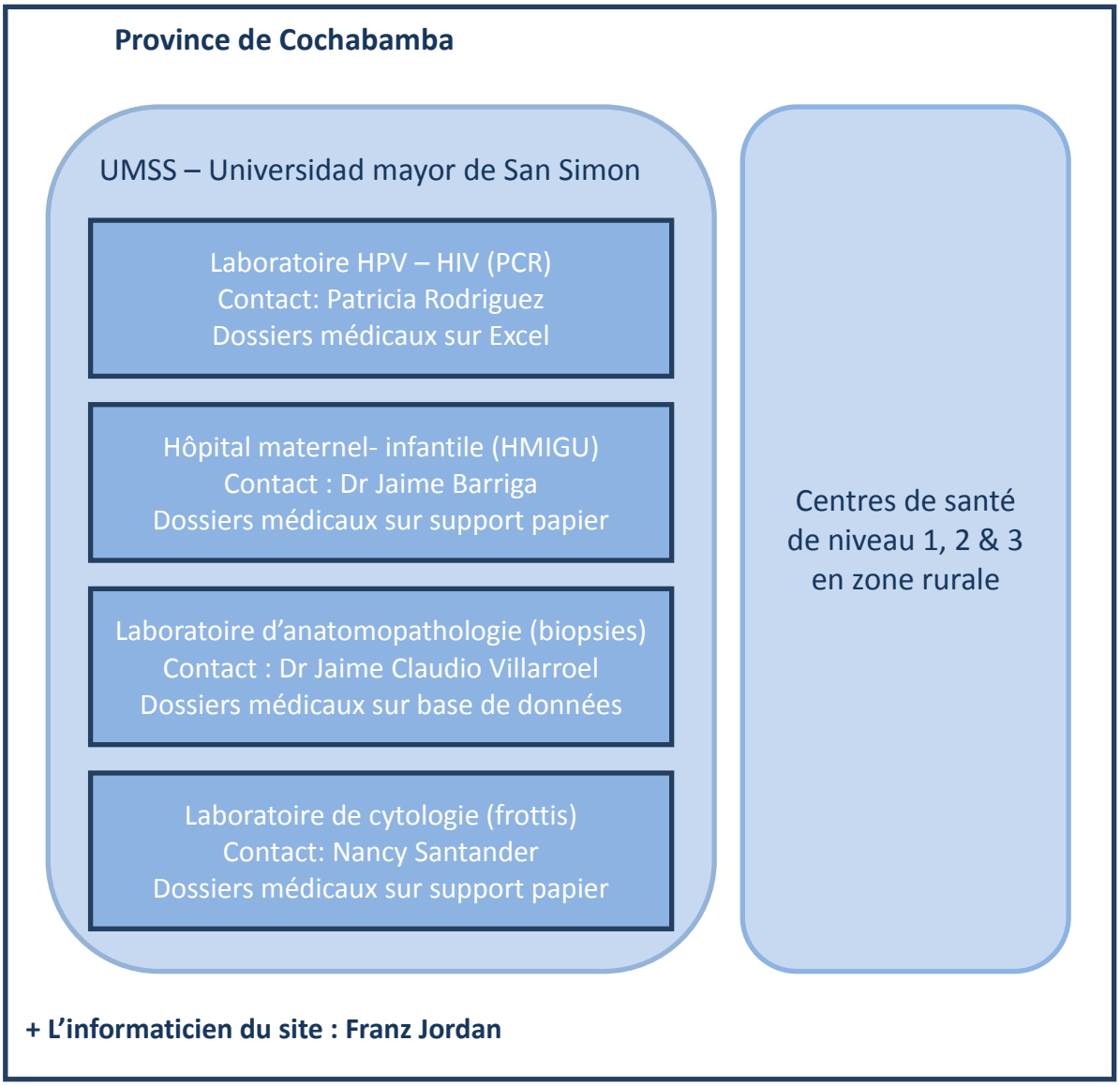


Figure 1: Plan du site Bolivien

En dehors du site universitaire, il existe des centres de santé de niveau 1 à 3 éparpillés dans la province de Cochabamba. Si une structure de niveau 3 correspond à un centre hospitalier muni de tous les équipements nécessaires et dans lequel travaillent médecins et personnel médical spécialisé, les infrastructures de niveau 1 et 2 ne disposent pas d'autant de ressources. Celles de niveau 2 reçoivent les services de médecins généralistes et spécialisés dans un centre de santé moins bien aménagé, tandis que celles de niveau 1 ne bénéficient pas toujours des compétences d'un médecin. Dans ces centres, une infirmière, ou même parfois un auxiliaire de santé, donne essentiellement des conseils médicaux et sert d'intermédiaire entre une patiente et un médecin[2].

1.3 Cahier des charges

Le rôle de l'Ecole polytechnique de Bruxelles, et plus particulièrement de la Cellule de coopération CoDePo, est de remédier aux divers problèmes de communication entre les différents intervenants. Cette révision passe par la création d'une base de données entièrement informatisée, ainsi que par une réévaluation de la logistique d'échange actuelle.

Afin de clarifier les objectifs du projet, de nombreuses discussions ont été réalisées avec le Professeur Véronique Fontaine, membre de la Faculté de Pharmacie de l'ULB, et les personnes de contact en Bolivie, en particulier avec Patricia Rodriguez. En effet, une grand partie du projet a consité à établir la liste des attentes de chacune des personnes concernées afin de répondre au mieux à la demande. Il a fallu de nombreux entretiens avec les acteurs du Sud pour mettre en place le cahier des charges suivant :

- Révision de la logistique actuelle concernant les stratégies employées dans le transfert des données ;
- Extraction de données provenant de fichiers Excel et de la base de données préexistants ;
- Création d'une base de données informatisée pour laquelle :
  - Un type de système de gestion de base de données devra être sélectionné ;
  - Un choix de langage de programmation efficace devra être effectué ;
  - Des fenêtres d'interface devront être mises en place afin d'assurer le lien entre la base de données et l'utilisateur ;
  - Un moyen de rendre la base de données accessible efficacement devra être déterminé ;
- Mise en place d'un code de programmation efficace et simplement modifiable par l'informaticien du site ou par des étudiants ultérieurement ;
- Recherche d'une technique permettant de respecter la confidentialité médecin-patient ;

[2] TAQUET, P. (Solidarité mondiale), *Santé en Bolivie : « pour les gens, l'avenir est une torture »*, http://www.solmond.be/pour-les-gens-l-avenir-est-une, <en ligne>, consulté le 19/02/2015.

- Exportation de certaines données de la base vers un tableau Excel pour, d'une part, les études statistiques réalisées par Véronique Fontaine et, d'autre part, le personnel médical désireux de pouvoir consulter sous format papier les informations relatives aux patientes actuellement suivies ;
- Création d'un manuel utilisateur pour former les médecins et personnel médical boliviens à l'utilisation correcte de l'application.

Un voyage de trois semaines aura lieu en Bolivie durant le mois de juillet 2015 dans le but d'installer la base de données. Lors de ce dernier, qui consistera en une première expérience de terrain, il faudra implémenter l'application, rectifier d'éventuels problèmes liés au système de gestion de la base de données, répondre à des demandes supplémentaires des locaux quant aux différentes fonctionnalités, et, finalement, initier le personnel médical à l'utilisation du programme.

2. Révision de la logistique actuelle

Actuellement, la situation est telle que tous les laboratoires du campus universitaire disposent d'au moins un ordinateur et une connexion internet est soit présente, soit en cours de mise en place. Les laboratoires ne disposant pas d'ordinateur à la base ont même demandé conseil à la Cellule de coopération de l'Ecole polytechnique de Bruxelles pour la sélection d'un matériel durable et de qualité. Les centres de santé ruraux n'ont, quant à eux, pas toujours accès à un ordinateur (qu'il soit professionnel ou personnel) et à une connexion internet. Par conséquent, un système hybride d'échange de données a été pensé. Les médecins de la faculté de médecine vont travailler via la base de données uniquement, tandis que les médecins et personnels infirmiers des centres de santé de niveau un à trois auront un système mixte. En effet, ils pourront continuer d'envoyer leurs éléments d'analyse avec une note manuscrite vers le campus universitaire. Une fois ces éléments testés dans les différents laboratoires, les résultats seront encodés dans la base de données par le personnel du laboratoire en question. Cette base de données étant accessible à tous les médecins des différents centres de santé, ceux-ci pourront consulter les résultats d'analyse peu importe l'endroit où ils se trouvent dans la province de Cochabamba. Toutefois, si un médecin ou une infirmière d'un centre de santé ne parvient pas à accéder ni à un ordinateur ni à internet, il reste la solution de l'export des données vers un tableau Excel. Celui-ci comportera toutes les données personnelles et principaux résultats d'examen relatifs aux patientes en cours de traitement. Ce document pourra être imprimé et envoyé sous format papier au médecin désireux de prendre connaissance du suivi de ses patientes. Un appel téléphonique vers les laboratoires et l'hôpital maternel-infantile du campus universitaire de l'UMMS est également une façon de récolter des informations particulières.

3. Système de gestion de base de données

La première grande décision qui a été prise concerne le choix d'un système de gestion de base de données. Cette section commence par un état de l'art détaillé avant d'aborder l'option sélectionnée. Cet état de l'art consiste en un état des lieux des techniques et logiciels utilisés de nos jours dans le domaine de la gestion de grandes bases de données.

3.1 Etat de l'art

Une base de données représente un ensemble volumineux de données structurées, les moins redondantes possibles. Celles-ci sont reliées entre elles et sont stockées sur des supports numériques centralisés ou distribués. Plusieurs utilisateurs et programmes travaillant en parallèle peuvent avoir accès à une même base de données. Ceci est, en effet, possible grâce à un Système de Gestion de Base de Données (SGBD), un logiciel qui prend en charge la structuration, le stockage, la mise à jour et la maintenance d'une base de données. Il s'agit de l'interface entre les informaticiens et les données, afin de définir des schémas et programmer des applications. D'un autre côté, il permet une communication serrée entre les utilisateurs et les données dans le but de consulter et mettre à jour ces données[3].

Les SGBD peuvent adopter différents modèles : hiérarchique, réseau, objet, … Ici, l'intérêt se porte en particulier sur le modèle relationnel (SGBDR), modèle mathématique basé sur la théorie des ensembles. Ce dernier structure les données en tables constituées de champs (colonnes) et d'entrées (lignes). Les champs représentent des attributs différenciés par leur nom et pouvant prendre différents types. Chaque entrée, quant à elle, est identifiée de manière unique par une clé primaire. D'autres clés dites étrangères, non nécessairement uniques, représentent la valeur de la clé primaire d'une table connexe et permettent ainsi d'établir des relations entre les tables. Parmi une série de SGBD, les plus connus sont sans doute MySQL, Oracle, PostgreSQL, Ingres, SQLite et Access. Il est à noter que MySQL et PostgreSQL sont libres et gratuits. Par ailleurs, MySQL est plus facile à utiliser et plus adapté aux petites bases de données que PostgreSQL, qui est lui orienté vers la gestion de bases de données massives (contenant des millions de milliards d'entrées).

La plupart des SGBDR utilise SQL (Structured Query Language) comme langage de programmation normalisé permettant à l'utilisateur de communiquer avec le système de gestion de base de données. Il s'agit d'un langage déclaratif qui permet, via l'utilisation de requêtes, de coder très simplement le résultat souhaité sans décrire la manière de l'obtenir.

Ces requêtes sont composées de mots clés permettant des manipulations simples des données : DELETE, INSERT, SELECT, UPDATE. D'autres commandes SQL, comme GRANT et

[3] SCENARI-PLATFORM, *BD et SGBD : vue d'ensemble*, http://scenari-platform.org/mobile-source/opale-demo/co/01_introduction_web/co/AC01vue.html , <en ligne>, consulté le 19/10/2014.

REVOKE, donnent l'opportunité de gérer les droits d'accès aux tables, alors que CREATE, ALTER et DROP permettent de créer des structures de bases de données[4].

Afin de gérer une base de données, l'installation d'un système de gestion de base de données relationnel est nécessaire. Deux types d'implémentation physique des SGBDR existent : certains SGBDR, comme SQLite et Access, utilisent un service de fichiers à partager afin d'accéder aux données et d'autres systèmes, tels que MySQL et Oracle, sont installés sur un serveur de données qui représente une application centralisée[5]. Cela signifie qu'en pratique, SQLite et Access s'utilisent très facilement : un fichier .sqlite ou .mdb (SQLite et Access, respectivement) représentant la base de données est placé sur l'ordinateur de l'utilisateur, et le programme SQLite ou Microsoft Access est utilisé pour accéder et modifier les données. Chaque utilisateur possède donc une copie de la base de données et la partager avec un autre se fait en copiant le fichier. Les SGBDR tels que MySQL, Oracle, PostgreSQL et Ingres nécessitent, quant à eux, la mise en place d'un serveur centralisé pour soutenir la base de données, auquel tous les utilisateurs se connecteront. Ainsi, les clients se connectant au même serveur voient la même base de données. Ce serveur doit être disponible en continu, jour et nuit, car la base de données est inaccessible si le serveur ne fonctionne pas. Si l'on décide d'utiliser de tels SGBDR, il faut installer le serveur quelque part, que ce soit sur l'ordinateur personnel ou sur un ordinateur séparé.

3.2 Choix

Tout d'abord, les systèmes de gestion de base de données que sont SQLite et Access ont été directement mis de côté, car leur implémentation en fichiers ne se prête pas facilement à l'échange d'informations. En outre, d'autres logiciels, comme Oracle pour ne citer que le plus connu, sont payants et ne peuvent donc pas être exploités en dehors de l'ULB, qui les met à disposition des étudiants. Leur utilisation en Bolivie dans un cadre médical engendrerait des contraintes légales et un hôpital ne peut pas prendre le risque d'avoir un problème de licence. Ces logiciels ont donc été également éliminés. Ensuite, le choix s'est restreint à MySQL ou à PostgreSQL, libres et gratuits. Toutefois, comme mentionné dans la section précédente, MySQL est plus répandu et plus facile à utiliser pour le même type de fonctionnalités. Par conséquent, MySQL 5.6 a été choisi pour supporter la base de données.

[4] MYSQL, *MySQL 5.6 Reference Manual*, http://dev.mysql.com/doc/refman/5.6/en/index.html , <en ligne>, consulté le 19/10/2014.
W3SCHOOLS.COM, *SQL,* http://www.w3schools.com/sql/sql_select.asp , <en ligne>, consulté le 18/10/2014.
DEVELOPPEZ.COM, *Le SQL de A à Z : 1ère partie – bases de données, SQL et types de données*, http://sqlpro.developpez.com/cours/sqlaz/fondements/#L5.1 , <en ligne>, consulté le 19/10/2014.
[5] DEVELOPPEZ.COM, *Le SQL de A à Z : 1ère partie – bases de données, SQL et types de données*, http://sqlpro.developpez.com/cours/sqlaz/fondements/#L5.1 , <en ligne>, consulté le 19/10/2014.

4. Langage de programmation

L'univers des langages de programmation est extrêmement vaste. Avant de déterminer lequel employer tout au long du projet, il a fallu réaliser un état des lieux du marché. De ce dernier découle la section suivante.

4.1 Etat de l'art

De nombreux langages de programmation peuvent servir à la conception d'applications graphiques accédant à une base de données, comme par exemple Python, C++, Java, Perl, Ruby, PHP, etc. Parmi cette gamme de choix, les options envisagées ont été celles correspondant à des langages enseignés durant le cursus du bachelier en ingénierie, ainsi que le PHP. Ces options sont succinctement passées en revue dans cette partie.

Python est un langage intuitif et donc facile à utiliser, de par sa programmation impérative, fonctionnelle et avec une gestion automatique de type « ramasse-miette» de la mémoire. Toutefois, il s'agit d'un langage interprété et non compilé comme certains. Ceci signifie que le programme effectue ses opérations de traitement et d'analyse à chaque exécution et non une seule fois à la compilation. Le désavantage principal de Python est donc que l'on perd beaucoup de temps à lancer le programme et à n'y voir qu'à ce moment-là les erreurs puisque la syntaxe, le nom des variables, le nom des fonctions et les types sont vérifiés à l'exécution. Un langage compilé a l'avantage, quant à lui, de détecter toutes les erreurs en une fois à la compilation et aucune ne peut échapper au programmeur car, s'il en reste, l'exécution du programme ne peut se produire.

Contrairement au Python, le C++ ainsi que Java sont tous deux des langages compilés, mais leur prise en main est plus longue. En particulier pour le C++, les nuances et spécificités du langage le rendent extrêmement ardu à manipuler dans la création d'interfaces, par rapport aux autres langages. Java est, quant à lui, plus simple à apprivoiser que le langage C++ dont il descend. Il est d'ailleurs très utilisé dans l'univers des bases de données et des applications graphiques.

Le langage PHP (pour Hypertext PrePocessor) est également un langage impératif et orienté objet. Cependant, contrairement à Python, il s'agit d'un langage de programmation chargé de coder des pages web dynamiques via un serveur http. En effet, le PHP peut, comme Python, fonctionner de manière locale, mais a l'avantage de pouvoir fonctionner également en application web[6].

[6] LE JOURNAL DU NET, *Comparatif : 14 langages au crible*, http://www.journaldunet.com/developpeur/tutoriel/out/040728-comparaison-langages1b.shtml, <en ligne>, consulté le 20/10/2014.

4.2 Choix

L'application a finalement été codée à l'aide du langage de programmation Python. Celui-ci a l'avantage d'être relativement simple à apprendre et a, en outre, été enseigné au cours du cursus au sein de l'Ecole polytechnique.

Java aurait également très bien pu être utilisé, puisqu'il constitue un excellent compromis entre les avantages et inconvénients de Python et C++. Cependant, il a été enseigné de manière plus sommaire durant le cursus. Quant à la solution web, bien qu'elle soit correcte et aussi facile à prendre en main que Python, aucun membre du groupe ne connaissait ni le langage PHP, ni les technologies du web par ailleurs. Sous ce terme se cachent énormément de notions indépendantes du PHP, comme celles de page web, HTML, CSS, Javascript, session, cookie, etc. Autant de connaissances de base qui auraient pris plusieurs mois à être maîtrisées.

Utiliser un langage déjà étudié auparavant consistait en un gain de temps non négligeable dans un projet de cette ampleur. L'apprentissage des notions de base de données et système de gestion de base de données a prôné sur l'initiation à un nouveau langage de programmation. La version 3.4 de Python a donc été exploitée.

5. Interfaces graphiques

Le lien entre l'utilisateur et la base de données se fait par l'intermédiaire des fenêtres d'interface. Ces dernières se doivent d'être intuitives pour faciliter l'apprentissage du personnel médical. En effet, des fenêtres bien faites peuvent amener un gain de temps non négligeable dans la compréhension du système, compte tenu du fait que le voyage ne durera que trois semaines.

5.1 Etat de l'art

Comme dit précédemment, les interfaces assurent la communication homme-machine. Le rôle d'une interface graphique est d'exposer à l'utilisateur des informations faciles à comprendre, tout en lui permettant de modifier ces informations à l'aide de divers champs de textes, listes, boutons et menus. Les GUI's (Graphical User Interfaces) fournissent les outils nécessaires à la mise en place de ces interfaces. Il en existe un grand nombre, mais l'un des GUI's les plus connus et facile à utiliser avec un programme en Python est Qt[7].

En effet, le logiciel Qt est non seulement facile à utiliser, mais permet également la création d'interfaces complètes et soignées. Il a l'avantage d'être portable et de présenter un look natif. Cela signifie que, sous Windows, ses fenêtres d'interfaces s'adapteront pour adopter un style « Windows », sous Mac, un graphisme propre à Mac, etc. De plus, Qt possède une bibliothèque de widgets très riche[8]. Ces différents arguments en ont fait un adversaire redoutable par rapport aux autres GUI's.

Il existe néanmoins d'autres GUI's compatibles avec Python. Parmi les plus connus, GTK+ tire également son épingle du jeu, mais il reste moins riche que Qt et plus difficile à apprendre. Bien qu'il soit portable sur les différents systèmes d'exploitation, il n'est que très peu harmonieux sur Mac ainsi que sur Windows[9]. Tkinter est, quant à lui, livré avec Python et uniquement compatible avec ce dernier. Cependant, ses fonctionnalités sont extrêmement limitées. Tkinter ne contient pas d'onglets, par exemple[10].

Les caractéristiques principales que devait gérer le GUI choisi étaient d'être utilisable avec le langage Python, d'être portable sur Windows, Mac et Linux et de contenir des librairies relativement bien fournies. C'est le cas de Qt et GTK+, il a donc fallu prendre une décision.

[7] PUISEUX, P., *Tutoriel sur PyQt : Présentation,* http://web.univ-pau.fr/~puiseux/enseignement/python/tuto-PyQt.01%28presentation%29.pdf, <en ligne>, consulté le 20/10/2014.
[8] Qt, *Qt Documentation : Qt Widgets*, http://doc.qt.io/qt-5/qtwidgets-index.html , <en ligne>, consulté le 21/10/2014.
[9] MONO, *GtkSharp*, http://www.mono-project.com/docs/gui/gtksharp/, <en ligne >, consulté le 21/10/2014.
[10] PYTHON, *Tkinter – Python interface to Tcl/Tk*, https://docs.python.org/2/library/tkinter.html, <en ligne >, consulté le 21/10/2014.

5.2 Choix

L'objectif était de construire des interfaces claires et intuitives, pour faciliter la formation du personnel médical. Qt, facile à utiliser en Python, disposant d'une grande collection de widgets et portable sous tous les OS principaux, est le logiciel qui répond le mieux à nos critères. C'est, en particulier, PyQt5 qui a été exploité (la dernière version en date). Le logiciel Qt designer, livré avec Qt, a permis de créer de telles fenêtres. Le principe de ce programme est de gérer de manière purement graphique le design des fenêtres d'application avant de convertir le fichier .ui et ainsi, obtenir le code Python correspondant à cette interface. Cette technique permet une approche stylistique simple et rapide, en comparaison avec une programmation classique.

Les différentes fenêtres d'interface de l'application, ainsi qu'une description précise de chacune d'elle, se trouvent en annexe dans la partie « Manuel utilisateur » (cf. Annexe D).

6. Mise en ligne

Les informations gérées par le système développé dans le cadre de ce projet doivent être disponibles en tout temps et en de nombreux endroits. Pour cela, les données doivent être stockées sur un réseau interne à l'UMSS ou sur Internet, et différentes alternatives ont été prises en compte.

6.1 Etat de l'art

Afin de mettre la base de données en ligne, deux options sont envisageables. Soit un serveur extérieur est utilisé, de manière à avoir en permanence accès à la base de données, soit un ordinateur doit être allumé en continu au sein de l'UMSS. La première option a un coût mensuel, mais ôte d'éventuels soucis techniques aux Boliviens. La deuxième alternative nécessite d'avoir un ordinateur relativement performant et, à chaque arrêt de la machine, la base de données sera momentanément inaccessible.

Si le choix se porte sur un serveur, il est nécessaire d'en choisir un qui supporte les bases de données MySQL. Pour cela, il faut sélectionner un serveur dédié (ou un serveur VPS pour *Virtual Private Server*). Un serveur dédié peut être vu comme un serveur informatique mis à disposition d'un seul utilisateur par un hébergeur. Sur ce serveur, il est possible de faire ce que l'on veut : installer MySQL, y avoir tous les droits, créer des utilisateurs,… A l'inverse, un hébergement mutualisé ne donne accès qu'à une base de données MySQL limitée, sur laquelle le stockage de données est permis, mais pas la création d'utilisateurs ni le changement de permissions. Un simple hébergement mutualisé basé sur PHP ne peut suffire, puisqu'il est nécessaire de contrôler totalement les permissions dans MySQL[11].

6.2 Choix

Après mûre réflexion, la décision a été de choisir un serveur VPS qui sera extérieur à l'hôpital. Cette solution a un coût, mais ce dernier reste très abordable pour le service offert.

Le choix du serveur a été fait d'après les critères suivants :

- La base de données serait accessible pour toute la province de Cochabamba dans un premier temps et pourrait facilement être appliquée au reste du pays si le système fonctionne bien;
- Ce système permet d'éviter de nombreux problèmes de maintenance liés à une erreur mécanique, humaine ou environnementale;

[11] NEXTADVISOR, *Dedicated vs. Shared Hosting: What's the difference?,* http://www.nextadvisor.com/blog/2014/02/06/dedicated-vs-shared-hosting-difference/, <en ligne>, consulté le 18/02/2015.

- Il n'y a pas besoin d'avoir un responsable base de données au sein de l'hôpital en cas de problème, ce qui évite de devoir avoir un membre du personnel chargé de réparer le système jour et nuit;
- Avec cette solution, la base serait accessible 365 jours par an.

Ensuite, la question a été de savoir si ce serveur serait hébergé en Bolivie ou ailleurs. De prime abord, les acteurs boliviens souhaitaient un serveur local, afin que leurs données restent à l'intérieur du pays. Les avantages et limites des deux options ont néanmoins été investigués, dans le but de faire un choix le plus objectif possible.

Bien qu'un serveur bolivien ait l'avantage d'être plus proche géographiquement parlant et que les informations ont tendance à transiter plus rapidement, il est très rapidement apparu qu'un serveur basé en Europe disposait de divers avantages non négligeables. Comme avantages nous pouvons citer la facilité administrative, la transparence des services, la limite des risques de corruptions ainsi que le coût. A titre d'exemple, le dernier serveur français en lisse coûtait 2 euros, face à 14 pour le bolivien. Sur le long terme, cette différence peut tout de même s'avérer non négligeable.

Finalement, en accord avec le Sud, le choix s'est porté sur un serveur français basé à Roubaix s'appelant OVH. Pour la somme de 1,99€/mois, la version Debian 7 en 64 bits du serveur VPS est active[12].

[12] Caractéristiques du serveur VPS acheté : https://www.ovh.com/fr/vps/vps-classic.xml

7. Importation de données et exportation vers Excel

Afin de ne pas perdre les données actuelles concernant les patientes atteintes du cancer du col de l'utérus et de pouvoir ainsi continuer d'assurer le suivi de ces patientes, une solution a dû être trouvée. Un transfert efficace des informations existantes vers la future base de données informatisée est effectivement indispensable.

De prime abord, après plusieurs entretiens avec les responsables des centres boliviens, les propositions suivantes avaient été avancées :

- Les données disponibles via un tableau Excel (données personnelles, gynécologiques et résultats du laboratoire HPV-HIV) seraient transférées directement sur la base de données grâce à un programme réalisé dans ce but. Celui-ci manipulerait un fichier Excel enregistré sous un format équivalent : le format CSV. En utilisant les propriétés de tels fichiers et en particulier la structuration particulière des données, ce morceau de code pourrait extraire facilement de l'information. Le format CSV (pour *Comma Separated Values*) consiste, comme son nom l'indique, en une division marquée par des virgules ou points-virgules. Ces fichiers sont de type « tableurs » : ils placent les données sur des lignes différentes en les scindant par à un caractère spécial de séparation. Les lignes représentent donc la première dimension, tandis que les virgules ou points-virgules permettent la subdivision en colonnes et amènent la deuxième dimension[13]. Un code basique correspondant à ce petit programme a été placé en annexe (cf. Annexe A). Il est utilisable uniquement à condition que la structure du tableau Excel soit exactement la même que celle de la base de données vers laquelle se fait le transit (mêmes noms de colonnes, même ordre des colonnes, mêmes options au sein des choix multiples,…).
- Les dossiers sur papier (certaines informations des centres de santé et les résultats d'examens effectués au laboratoire de cytologie) devraient être encodés manuellement une fois la base de données installée en Bolivie.
- N'ayant pas de contact privilégié avec le laboratoire d'anatomopathologie qui détient déjà les résultats d'examen des patientes dans une base de données informatisée propre, aucune information concrète à propos de ce stockage des données n'a pu être communiquée. Aucun programme n'a donc pu être codé afin de transférer les informations d'une base de données à une autre. Il a donc été conclu, en accord avec les contacts au Sud, qu'un encodage manuel serait également indispensable.

Cependant, au fil du projet, nous avons réalisé que la première option mentionnée ci-dessus était compliquée à mettre en place. En effet, pour que cette solution soit applicable, il aurait fallu pouvoir acquérir des Boliviens un tableau Excel uniformisé et immuable pour le

[13] EDOCEO, *Comma Separated Values (CSV) Standard File Format*, http://edoceo.com/utilitas/csv-file-format, <en ligne>, consulté le 5/11/2014.

laboratoire HPV-HIV, ainsi que pour les informations gynécologiques. Ceci n'a pas su être obtenu à temps. Les tests et leur encodage n'ont fait que varier tout au long du projet et, bien que le code soit fonctionnel, il était délicat d'avancer sur ce terrain. Par ailleurs, il ne s'agissait que du transfert d'une partie seulement des données relatives aux patientes, les autres nécessitant dans tous les cas d'un ré-encodage. La priorité n'a donc pas été portée sur le développement de cette fonctionnalité. En accord avec les acteurs du Sud, les informations présentes sur fichiers Excel devront par conséquent être ré-encodées manuellement dans la base de données. Gardons néanmoins à l'esprit qu'une importation automatique des données est encore possible, dans le cas où les médecins Boliviens parviennent à formater correctement leurs données sous forme d'un fichier Excel adéquat, d'ici à notre venue sur le terrain.

Le cahier des charges spécifiait qu'afin d'aider le Professeur Véronique Fontaine dans ses statistiques et recherches, certaines données devaient pouvoir être extraites de la base de données vers un tableau Excel. De plus, l'export vers Excel devait également permettre aux médecins d'obtenir sous format papier quelques informations cruciales qui résumeraient l'état de santé des patientes actuellement suivies. Cette partie, quant à elle, a pu être menée à bien. Après avoir convenu des informations nécessaires à l'élaboration de ces statistiques et de ces dossiers papiers, les données de la base ont été mises sous format CSV suivant le phénomène inverse à celui expliqué ci-dessus, avant d'ouvrir ces fichiers CSV avec Excel.

8. Implémentation de l'application

La sélection et les différentes options ayant été abordées et expliquées en détails, la programmation à proprement parler peut à présent être plus amplement analysée. Le code associé à notre base a été placé en annexe, muni de nombreux commentaires pour aider à la compréhension. Cependant, certaines remarques quant à la structure générale choisie doivent être réalisées.

8.1 La classe AbsractList

La base de données a été pensée de manière à être, d'une part, la plus concise au niveau du code et, d'autre part, la plus souple et modulable possible dans une logique de pérennité du projet. En effet, il est aisément envisageable que la liste des examens pour un laboratoire soit modifiée en fonction des avancées scientifiques et technologiques. Il est également tout-à-fait concevable qu'à terme, de nouveaux laboratoires soient créés dans la liste qui leur est dédiée, que le nom de certains laboratoires change et que d'autres ferment leurs portes. De la même manière, la liste des patientes est supposée changer en permanence avec l'arrivée de nouvelles femmes. Enfin, lors de l'ajout d'un nouvel utilisateur de la base de données ou de l'attribution de nouveaux privilèges à un utilisateur donné, des modifications de la liste des médecins ont lieu. Après réflexion, il s'est avéré que les opérations à effectuer sur ces différentes listes (PatientList, UserList, LaboList et TestList) sont tout-à-fait similaires : il s'agit à chaque fois d'ajouter, de modifier ou de supprimer un élément d'une liste. De là, l'idée est venue de créer un système le plus général possible qui permettrait d'effectuer les mêmes opérations sur ces quatre catégories de listes.

Ceci s'est traduit par l'apparition d'une classe abstraite au niveau de la programmation. Ci-dessous, l'UML associé a été représenté avec ses attributs et méthodes pour plus de clarté.

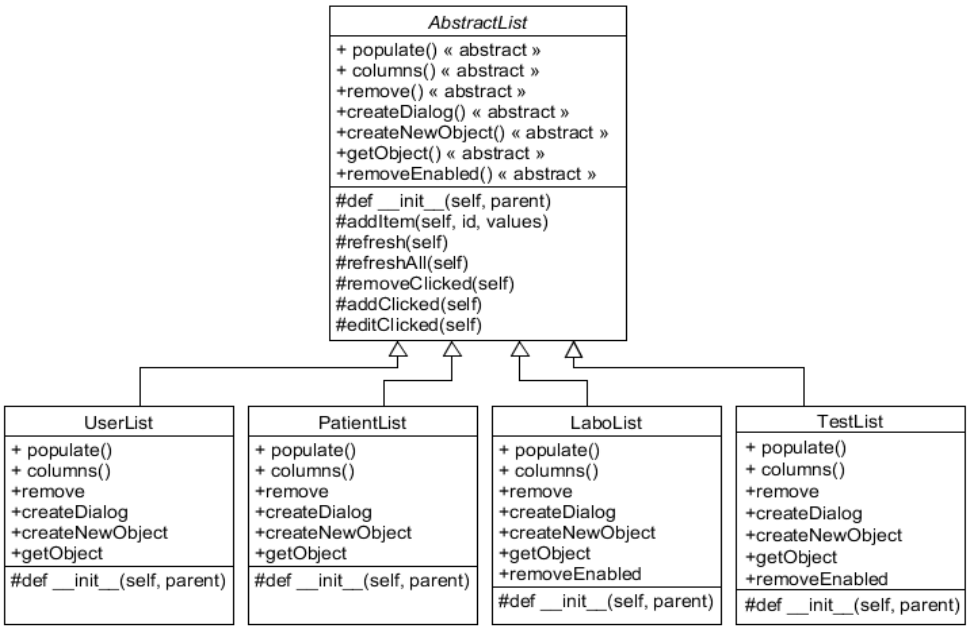


Figure 2 : Diagramme UML associé à la classe AbstractList et à ses classes filles

Pour rappel, le rôle de ces différentes fonctions et les interactions détaillées ont été placées dans les commentaires du code en annexe (cf. Annexe C).

8.2 Modèle-vue-contrôleur

La structure générale décrite au point précédent implique une certaine redondance dans la création des fichiers. Pour une fenêtre nommée X :

- X.ui représente le fichier Qt Designer de la fenêtre.
- ui_X.py est le fichier Python généré à partir de X.ui. Il contient la classe Ui_X.
- X.py contient le code Python pour la fenêtre. Ce fichier contient une classe X, qui hérite de QWidget, QDialog ou QMainWindow. Il comporte également une ligne « from ui_X import* », afin de rendre Ui_X utilisable par X. Un tel fichier X.py peut, par exemple, renfermer la définition du slot (fonction appelée lorsqu'un événement s'est produit) d'un bouton composant la fenêtre. Ce même bouton aura été déclaré dans le fichier ui_X.py.

Cette découpe s'accorde pour le mieux avec la règle de bonne pratique en Python, qui consiste à garder les fichiers les plus concis et simples possibles. Afin de respecter cette convenance, les programmeurs Python ont recours à de nombreuses découpes en fichiers plus petits jusqu'à n'obtenir que des fichiers .py à fonction unique. La structure du code permet cette division. En effet, les fichiers de type « ui_X.py » créent des widgets et ne font que cela, tandis que les fichiers « X.py » contiennent les setters et getters, ainsi que la gestion des signaux et slots d'une fenêtre.

En informatique, une architecture bien connue des programmeurs permet d'organiser les interfaces graphiques au sein d'un programme. Cette dernière porte le nom d'architecture *modèle-vue-contrôleur*. Ce type de structure consiste à diviser le code en trois entités distinctes qui sont, comme le nom l'indique : le modèle, la vue et le contrôleur.

Le *modèle* manipule les données utilisées par le programme. C'est, par conséquent, la partie qui s'interface avec la base de données. Son rôle consiste à gérer une grande quantité de données, tout en garantissant leur intégrité au cours des opérations et actions qui leur seront appliquées. Le modèle, via les requêtes qu'il comporte, offre des méthodes pour l'accès aux données et la mise à jour.

Le lien entre l'utilisateur et la base de données est assuré par la *vue*. La fonction de cette dernière est, d'une part, d'afficher les données récupérées auprès du modèle et, d'autre part, de recevoir des commandes de l'utilisateur.

Les différents événements perçus par la vue sont envoyés au *contrôleur* qui va orchestrer le tout. Il va se charger de la synchronisation du modèle et de la vue. En effet, il est informé des manipulations provenant des utilisateurs et va provoquer les actions qui en découlent. Dans le cas où l'action engendre une modification de données par le modèle, c'est également le contrôleur qui effectue la demande de mise-à-jour à la vue[14].

Le schéma suivant clarifie les interactions entre ces trois entités. Comme on peut le voir, dans le cas où un utilisateur (l'administrateur de la base de données dans cet exemple) effectue une demande, cette dernière est reçue par le contrôleur qui va traiter les données du modèle avant de les communiquer à la vue qui va, à son tour, les représenter à l'utilisateur. Le diagramme ci-dessous illustre la construction MVC employée pour la gestion de la boîte de dialogue utilisateur, mais la logique est la même dans le cas des laboratoires, des tests ainsi que des patientes. La structure se répète donc pour ces différents cas.

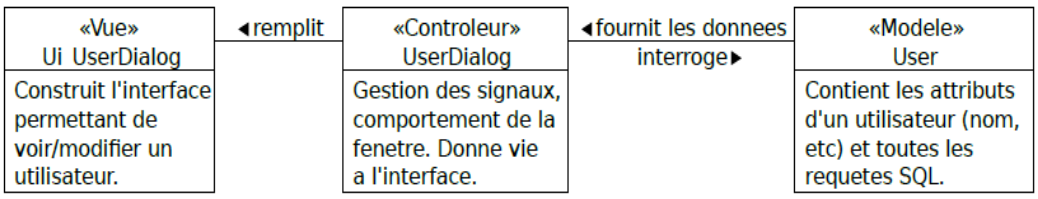


Figure 3: Interactions Modèle - Vue - Contrôleur

Cette architecture contient un grand nombre de fichiers courts et à fonction unique, ce qui permet, dans le cas d'une modification, de ne toucher qu'à très peu voire à un seul fichier. Ceci empêche l'apparition de nombreux bugs lors de la génération de nouveau code.

[14] YUNES, J.-B., *Interfaces graphiques (GUI)*, http://www.liafa.univ-paris-diderot.fr/~carton/Enseignement/InterfacesGraphiques/MasterInfo/Cours/Swing/mvc.html, <en ligne>, consulté le 14/02/15

Dans le cas où des erreurs se manifestent malgré tout, il est chose aisée de trouver le fichier à l'origine du problème.

9. Sécurisation du réseau

Cette section est consacrée à la manière dont les informations sont sécurisées dans la base de données. Un dispositif de sécurité est indispensable pour garantir la confidentialité des données personnelles inhérentes à chacune des patientes. Cette problématique étant assez dense et complexe, un chapitre lui a été entièrement dédié.

9.1 Prérequis

Avant de passer à la sécurisation en tant que telle, il est indispensable de définir les acteurs et principales applications mises en jeu. Cette sous-section a pour but de clarifier toutes ces notions.

En informatique, un *réseau* peut être défini comme un ensemble de machines connectées entre elles et qui échangent des informations via des flux de données. Cette notion reste cependant très générale puisqu'elle définit aussi bien l'ensemble des machines que le protocole de communication qui les lient ou encore, la connexion entre les différentes machines[15].

Dans le cas du réseau informatique, deux acteurs interviennent dans le processus de communication. Le programme qui initie la communication est appelé *client* et celui qui initie la connexion n'est autre que le *serveur*.

Dans le contexte de la base de données, MySQL fait office de serveur, tandis que le rôle de client est joué par tout utilisateur s'adressant à ce serveur dans le but d'établir une connexion. Pour que cela fonctionne, il faut toutefois que serveur et client parlent le même langage, sans quoi, la communication ne peut se faire correctement. Cela nous amène à la prochaine sous-section.

9.2 Passage de Python à MySQL

Comme dit précédemment, c'est MySQL qui joue le rôle de serveur. Cependant, comme ce dernier contient des informations potentiellement confidentielles, tout utilisateur ne peut y avoir accès. MySQL va donc obliger le client à d'abord décliner son identité et accordera, ensuite, le droit ou pas à ce dernier de se connecter et avoir accès à la base de données.

Dans le cadre du projet, d'une part, MySQL a pour rôle de stocker les données des patientes et d'autre part, un programme codé en Python se comporte comme un client en accédant au serveur pour récupérer ou traiter des données. Cependant, ces deux

[15] DIDIER, J.- Y., *Introduction au réseau,* http://lsc.univ-evry.fr/~didier/webpage/pedagogie/ii25_final.pdf, <en ligne>, consulté le 14/02/15.

programmes ne parlent pas le même langage, puisque MySQL est codé en C. Il va donc falloir glisser un « interprète » entre les deux.

Le driver Python choisi pour assurer ce rôle de « glue » n'est autre que PyMySQL. Les raisons de ce choix sont très simples : PyMySQL est codé en Python pur, est rapide, est compatible avec SSL et existe depuis longtemps. Aucun interprète supplémentaire n'est nécessaire entre l'application Python et le serveur MySQL, car le driver sélectionné peut communiquer directement avec MySQL tout en étant compréhensible par le programme Python de par sa programmation dans le même langage. De nombreux autres drivers Python capables de lier directement une application Python à un serveur MySQL existent. Cependant, avant toute chose, les alternatives qui ne supportent que Python2, comme MySQL for Python, mxODBC, pyodbc et bien d'autres encore, ont dû être rejetées. En effet, l'application créée dans le cadre de ce projet est codée en Python 3.4. Les autres options possibles, telles que mypysql ou encore MySQL Connector/Python, ne présentent jamais en même temps les quatre propriétés que remplit PyMySQL, en particulier le fait d'être codé en Python pur. La plupart de ces alternatives est plutôt codée en C, mais possèdent un petit morceau de code en Python, qui se charge de jouer le rôle de « glue » entre les deux mondes. Le désavantage du code C est qu'il est compilé dans un fichier .dll, et ce fichier doit être distribué. A l'inverse, le code Python, dans des fichiers Python, peut se distribuer tel quel et est donc facile à utiliser et à installer[16].

Le schéma ci-dessous a pour but de clarifier la position de chaque intervenant. Le serveur MySQL est installé sur un serveur dédié, tandis que PyMySQL est un module utilisé par le code Python de l'application. PyMySQL fait donc partie de l'application et sera distribué avec elle.

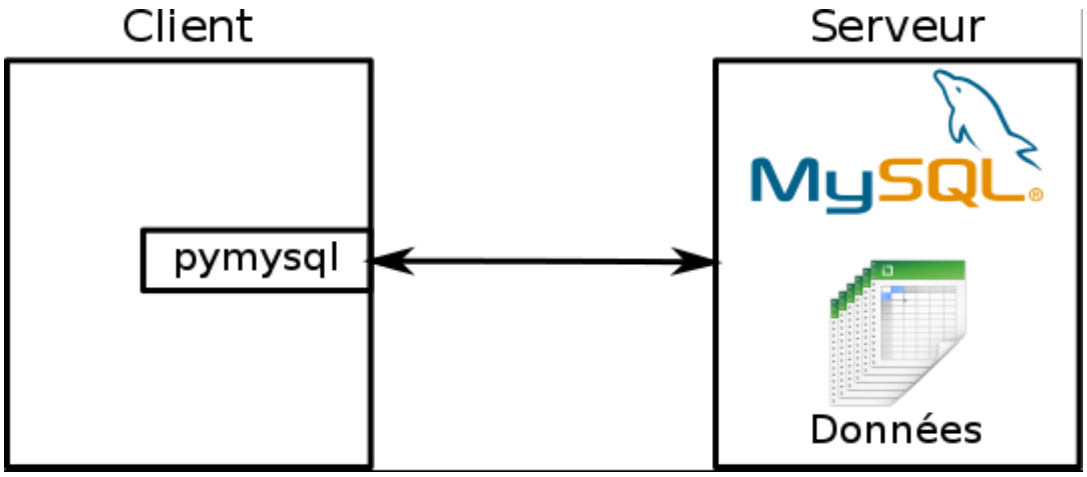


Figure 4 : Passage de Python à MySQL et inversement

[16]PYTHON, *MySQL*, https://wiki.python.org/moin/MySQL, <en ligne>, consulté le 21/10/2014.

9.3 Chiffrer les connexions

Lorsque deux programmes communiquent entre eux sans être sur un même ordinateur, les données vont devoir être envoyées d'un ordinateur à l'autre. Le trajet que vont suivre les informations est généralement long et fait intervenir plusieurs acteurs et programmes. Par conséquent, durant cet échange de données via le réseau internet, un programme pirate externe pourrait très bien capter des informations personnelles sensibles concernant des patientes ou encore, l'identifiant et mot de passe d'un utilisateur. En effet, tout peut être observé sur internet. Dans le cadre d'une base de données comportant des informations médicales, il est totalement inenvisageable de prendre le risque qu'une telle chose se produise. Pour éviter que le flux de données soit interprété et lu par quelqu'un qui n'en aurait pas l'autorisation, il a fallu chiffrer les données. Ce chiffrement porte autant sur les mots de passe et identifiants des utilisateurs que sur les données médicales des patientes.

Bien que la création d'un cryptage performant ait été un casse-tête pour les mathématiciens de l'époque, il est dorénavant assez simple de chiffrer nos données. Ce n'est autre que la technologie SSL (*Secure Socket Layer*) qui a été exploitée pour sécurisée la transmission des données sur Internet. SSL chiffre de manière asymétrique et protège les données transmises. Pour activer le chiffrement, il suffit d'une configuration rapide du serveur dédié. Si l'on se rappelle la structure mentionnée à la section précédente, c'est PyMySQL qui se charge de la communication entre un client et un serveur MySQL. C'est donc au niveau de PyMySQL que se fait le cryptage. Lors de l'envoi de données vers un serveur MySQL, PyMySQL chiffre les données à l'aide de la clé publique correspondant au serveur MySQL avec lequel il souhaite communiquer, clé envoyée par ce même serveur. Ce dernier n'a qu'à déchiffrer les informations reçues à l'aide de sa clé privée. A l'inverse, si MySQL désire transférer des données au client, il utilise la clé publique du client, générée par PyMySQL, pour chiffrer les données. Seul le client peut les déchiffrer avec sa clé privée. Toutefois, l'authentification du serveur est également nécessaire. Celle-ci consiste à s'assurer que le serveur MySQL qui se présente à PyMySQL est bien celui qu'il prétend être. Elle est rendue possible grâce à l'intégration de la ligne de code suivante au niveau de l'application Python :

```
pymysql.connect(username=..., password=..., ssl = {
    'ca' : 'mysql_ca_cert.pem'})
```

Ce morceau de code permet l'envoi d'un fichier .pem à PyMySQL. Ce fichier contient des informations permettant à PyMySQL de vérifier qu'il s'agit bien de la clé publique du serveur MySQL « codepo » qu'il reçoit et pas celle d'un autre serveur.

Pour résumer, la technologie SSL joue sur tous les fronts : elle gère la confidentialité (via le chiffrement asymétrique RSA), l'authentification (avec le fichier .pem) et l'intégrité de

l'information échangée (via l'utilisation des hash en interne)[17].

[17] CIUBOTARU, B., et al. "Advanced Network Programming – Principles and Techniques", Springer, Londres, 2013, Pages 89-100.

10. Gestion de la confidentialité

Lors de la confection d'une base de données, à côté de la sécurisation du réseau, la gestion de la confidentialité au sein du corps médical est un aspect primordial à prendre en compte. Seuls des médecins reconnus comme fiables peuvent avoir accès aux données des patientes et ceux-ci ne peuvent modifier que les résultats des examens qu'ils pratiquent. Une longue réflexion au sujet des différentes options a eu comme finalité la sélection d'une méthode simple et fiable. Cette dernière consiste à utiliser directement les propriétés inhérentes aux tables de MySQL. Cette section est consacrée à la mise en place d'un système de protection des données personnelles des patientes.

10.1 Les privilèges

La base de données contient six tables : « patientes », « resultados », « laboratorios », « examenes », « usuarios » et « grupos » (cf. Annexe B).

- La table « patientes » n'est autre qu'un tableau dans laquelle sont stockées toutes les informations personnelles des patientes.
- Quant à la table « usuarios », elle est complétée lors de l'inscription d'un membre du personnel médical à la base de données.
- Les différents laboratoires et leurs caractéristiques sont repris dans la table « laboratorios ».
- La table « examenes » reprend les différents examens effectués dans les différents laboratoires.
- Les résultats de ces examens se trouvent dans une autre table appelée « resultados ».
- Les différents privilèges pouvant être accordés à un médecin sont définis au sein de la table « grupos ».

La table « grupos » étant la plus pertinente à détailler afin d'expliquer la gestion de la confidentialité, celle-ci a été représentée ci-dessous sous un format plus didactique que la table MySQL brute (cf. Figure 5). Elle a été construite sur base de nombreux échanges avec les contacts boliviens. Un numéro de groupe est assigné à chaque combinaison de privilèges qu'un membre du corps médical peut se voir attribuer. Au total, 16 groupes ont pu être établis.

Pour résumer, un médecin lambda travaillant au sein d'un certain laboratoire peut voir tous les résultats d'examens de tous les laboratoires, ainsi que les données gynécologiques et personnelles des patientes. Cependant, ce médecin pourra, en général, uniquement apporter des modifications aux résultats des tests effectués dans son laboratoire. Ce n'est pas le cas du personnel du laboratoire HPV-HIV et des gynécologues et médecins des centres de santé de niveaux 1, 2 et 3. Ils peuvent, quant à eux, également modifier les données personnelles des patientes, car ce sont les principales personnes en charge du suivi médical. Par ailleurs, les médecins du laboratoire HPV-HIV ont également la possibilité d'encoder de

nouvelles informations relatives aux tests réalisés au sein du laboratoire d'anatomopathologie. En effet, il se pourrait que, dans un premier temps, le service d'anatomopathologie soit un peu retissant quant à l'utilisation d'une nouvelle base de données commune à tous, étant donné qu'ils en disposent déjà d'une, propre à leur laboratoire. Pour pouvoir effectuer cette transition tout en douceur, il a donc été proposé que le laboratoire de Patricia Rodriguez débute l'encodage des données, le temps que le laboratoire d'anatomopathologie prenne ses marques avec le projet présenté.

En outre, afin de pouvoir assurer la pérennité du projet, un responsable par laboratoire sera autorisé à modifier les tests de son laboratoire. Enfin, un administrateur aura le droit d'ajouter ou non des utilisateurs à la base en précisant à chaque fois leurs privilèges. C'est effectivement cette personne qui pourra déterminer si oui ou non la personne désireuse de s'inscrire est assez fiable pour avoir accès aux informations relatives aux patientes. L'administrateur doit donc bien connaître le personnel médical impliqué dans le dépistage. Pour le moment, les administrateurs en charge sont au nombre de deux : Patricia Rodriguez du laboratoire HPV-HIV et J. Aillende, gynécologue doctorant qui travaille sous les directives du docteur Barriga. L'administrateur sera aussi en charge de la mise à jour de la base de données : elle devra supprimer les médecins à la retraite ou encore, décédés.

	Grupo	Modifier les données personnelles	Modifier les données cliniques	Modifier les données HPV-HIV	Modifier les données cytologie	Modifier les données anatomopathologie	Gérer les utilisateurs	Modifier les tests
Personnel de santé 1	**1**	1	1	0	0	0	1	1
HPV-HIV 1	**2**	1	0	1	0	1	1	1
Cytologie 1	**3**	0	0	0	1	0	1	1
Anatomopathologie 1	**4**	0	0	0	0	1	1	1
Personnel de santé 2	**5**	1	1	0	0	0	1	0
HPV-HIV 2	**6**	1	0	1	0	1	1	0
Cytologie 2	**7**	0	0	0	1	0	1	0
Anatomopathologie 2	**8**	0	0	0	0	1	1	0
Personnel de santé 3	**9**	1	1	0	0	0	0	1
HPV-HIV 3	**10**	1	0	1	0	1	0	1
Cytologie 3	**11**	0	0	0	1	0	0	1
Anatomopathologie 3	**12**	0	0	0	0	1	0	1
Personnel de santé 4	**13**	1	1	0	0	0	0	0
HPV-HIV 4	**14**	1	0	1	0	1	0	0
Cytologie 4	**15**	0	0	0	1	0	0	0
Anatomopathologie 4	**16**	0	0	0	0	1	0	0

Figure 5 : Table "grupos" décrivant les privilèges

Il est à noter que cette table suit un modèle booléen. En effet, elle n'est composée que de 1 (pour marquer une action permise) et de 0 (pour marquer une action interdite).

10.2 Création d'un nouveau compte utilisateur

Lorsqu'un membre du personnel médical en charge du suivi des patientes atteintes du cancer du col de l'utérus souhaite se créer un compte pour accéder à la base de données, il doit contacter un des administrateurs. Si l'administrateur décide d'autoriser l'accès à la base de données à ce nouvel utilisateur, il lui crée un compte en complétant, via une interface, les champs « Nombre » (nom), « Apellido » (prénom) et « Célula de identidad » (numéro de registre national) de la table « usuarios » (cf. Figure 4). Les autres colonnes de la table « usuarios » sont laissées vides. Un autre paramètre dont l'administrateur doit entrer la valeur dans l'interface de création d'un utilisateur est le mot de passe (« Constraseña »). Pour une raison de sécurité, le mot de passe n'est évidemment pas stocké dans la table « usuarios », mais est seulement affiché dans l'interface de l'administrateur au moment de l'ajout d'un nouvel utilisateur. Un bouton de l'interface de création permet de créer un mot de passe provisoire généré pseudo-aléatoirement par un algorithme. Au niveau du serveur MySQL, ce mot de passe est passé dans une fonction de hachage cryptographique et le hash obtenu est stocké dans la table « users » de MySQl (table détaillée dans la section 9.4.4).

Parallèlement à cela, l'administrateur transmet (par mail, poste, téléphone, oralement, etc) à l'utilisateur son mot de passe provisoire (autrement dit, celui fourni de manière pseudo-aléatoire par un algorithme) et précise que son identifiant n'est autre que son numéro de registre national. À partir de ce moment, l'utilisateur peut accéder au système en utilisant son login et le mot de passe qu'il a reçu. Il fait partie d'un groupe choisi par l'administrateur, et il ne peut pas en changer lui-même. Cependant, si l'utilisateur le désire, il a la possibilité de changer de mot de passe à tout moment en complétant le champ « Cambiar contraseña » présent dans la fenêtre « Añadir o modificar los datos personales » (cf. Manuel d'utilisation dans l'Annexe D). Ce nouveau mot de passe repassera dans la fonction de hachage et le hash correspondant à ce médecin dans la table « users » de MySQL sera modifié.

Au sein de cette même fenêtre « Añadir o modificar los datos personales », l'administrateur invite également l'utilisateur à compléter ses informations personnelles (date de naissance, téléphone, adresse e-mail et domicile) lors de sa première connexion à la base de données. Une fois encodées, ces données rempliront les colonnes vides de l'entrée (la ligne) correspondant à l'utilisateur dans la table « usuarios ». Cet encodage des données relatives au médecin bénéficiaire de la base de données par le médecin lui-même permettra de réduire le travail de l'administrateur et ces informations seront utiles pour l'affichage. Par exemple, une fonctionnalité à laquelle il a été pensé mais qui n'a pas pu voir le jour par manque de temps serait de pouvoir lire à côté d'informations encodées : "analyse faite par Dr. Rodriguez Patricia, le XX/XX/XX". Avec la structure actuelle du code, cette amélioration serait facilement réalisable lors de l'implémentation de la base de données en juillet 2015 ou par une nouvelle équipe d'étudiants ayant pour projet d'ajouter des fonctionnalités supplémentaires à la base de données.

	ID	Apellido	Nombre	Nacimiento	Célula de identidad	Grupo	Telefono	Email	Domicilio
Usuario 1									
Usuario 2									
Usuario 3									

Figure 6 : Table "usuarios"

10.3 Privilèges au sein de l'application Python

Le code Python de l'application permet de différencier les 16 combinaisons de privilèges possibles détaillées plus haut. En effet, parmi les attributs d'un utilisateur, on retrouve son numéro de groupe qui a été récupéré de la table « usuarios » de la base de données MySQL « codepo ». Au sein du programme Python, se trouve une série de boucles de condition if, qui testent la valeur de cet attribut et y réagissent en cachant certains onglets d'interface et en rendant inchangeables les résultats de certains laboratoires. Il s'agit donc là uniquement d'un lien entre le numéro de groupe (ou encore les privilèges) et les interfaces graphiques. Certaines données ne peuvent pas être modifiées au niveau des fenêtres d'interface, mais rien n'empêche une altération des données au niveau de la base de données présente dans MySQL. Cette réflexion nous amène au point suivant.

10.4 Permissions dans MySQL

MySQL est un serveur très complet qui permet de stocker des données ainsi que de gérer leur accès. Comme vu plus haut, quand on se connecte à MySQL, on doit lui fournir un nom d'utilisateur et un mot de passe.

En réalité, MySQL dispose d'une liste d'utilisateurs globaux à tout MySQL : la table « users ». Cela signifie qu'il n'y a pas que les utilisateurs de la base de données « codepo » dans la liste, mais chacun des utilisateurs ayant un nom et un mot de passe qui permettent de le renvoyer vers la base de données dans laquelle il est enregistré. Quand quelqu'un se connecte, MySQL demande un nom d'utilisateur et un mot de passe, passe le mot de passe dans une fonction de hachage cryptographique et vérifie dans sa liste qu'un utilisateur avec de tels nom et hash existe. Si c'est le cas, le client est accepté. Sinon, MySQL lui raccroche au nez.

Une fois qu'un utilisateur a été reconduit vers sa base de données, il peut ou non effectuer des actions sur celle-ci en fonction des privilèges que lui a accordés l'administrateur de la base de données. En effet, en passant par des requêtes SQL (GRANT et REVOKE entre autres), celui-ci est habilité à définir différents droits aux utilisateurs.

Par exemple, si un utilisateur "Patricia" a le droit de lire et modifier les données

personnelles des patientes (table « pacientes ») au sein de la base de données, la requête sera la suivante :

GRANT select, update ON pacientes TO “Patricia”;

Il est primordial de noter que, par défaut, un utilisateur n’a aucun droit. Par conséquent, cet utilisateur n’a pas le droit de supprimer des informations, par exemple.

Dans le cadre du projet, cette notion de permissions au sein de MySQL a été exploitée. En effet, malgré la gestion des groupes de privilèges au sein de l’application Python détaillée plus haut, la sécurisation n’est pas optimale. Il suffirait qu’un médecin avec quelques notions d’informatique, muni de l’adresse IP du serveur VPS écrite dans l’exécutable (cf. section 11) et de ses identifiant et mot de passe attribués par l’administrateur, utilise la commande « mysql » pour se connecter au serveur et changer ses privilèges ou ceux d’autres médecins. Il pourrait, par exemple, s’octroyer la fonction d’administrateur. Les privilèges MySQL permettent d’éviter de rencontrer ce genre de situations.

Dans le cas particulier du projet Codepo, deux principaux types de privilèges ont été définis au sein même du serveur MySQL. Etant donné que les permissions MySQL ne peuvent pas s’appliquer sur une partie seulement d’une table et que les résultats de tous les laboratoires confondus se trouvent tous au sein de la table « resultados », il n’a pas été possible de différencier à ce niveau les privilèges de deux médecins provenant de deux laboratoires différents. Par conséquent, dans MySQL, il a été défini de base que tous les utilisateurs peuvent voir et modifier tous les résultats des tests. A cela s’ajoute deux privilèges possibles : l’administration de la base de données (ajout, suppression ou modification d’utilisateurs) et l’administration d’un laboratoire (ajout, suppression ou modification de tests du laboratoire en question). La combinaison de ces deux permissions donne lieu à 5 groupes de privilèges au niveau du serveur MySQL.

Lorsque les permissions d’un utilisateur seront modifiées dans la boite de dialogue du programme, son numéro de groupe de privilèges sera modifié dans la table « usuarios ». Ensuite, MySQL sera également informée de ces changements de permission, pour que les permissions vues par MySQL correspondent toujours à la version la plus récente des permissions que possède l’utilisateur.

Pour résumer, dans le cadre du projet, une tentative d’extraction du meilleur des deux mondes a été opérée :

- Le programme « codepo » gère les permissions exactes des utilisateurs;
- MySQL connait toutes les permissions de l'utilisateur et les vérifie également. Un utilisateur qui modifie le programme ou essaie de contourner ses priorités buttera

donc contre les protections inamovibles de MySQL et ne pourra pas toucher aux données qu'il n'a pas le droit de modifier.

10.5 Architecture de l'application

L'architecture de l'application possède une caractéristique intéressante du point de vue de la sécurité des accès à la base de données. Comme décrit sur le schéma ci-dessous, le programme va en premier lieu vérifier l'identité de l'utilisateur avant d'ouvrir une connexion à la base de données. Dans la plupart des applications, le processus inverse se produit (c'est-à-dire que le programme va commencer par se connecter à MySQL avant de se lier à l'utilisateur) et il est important de noter cette spécificité[18].

Cette architecture permet de rentabiliser les propriétés inhérentes à MySQL en utilisant un maximum les permissions et utilisateurs en son sein. En effet, de cette manière, les permissions sont vérifiées dans le programme, ainsi que dans MySQL.

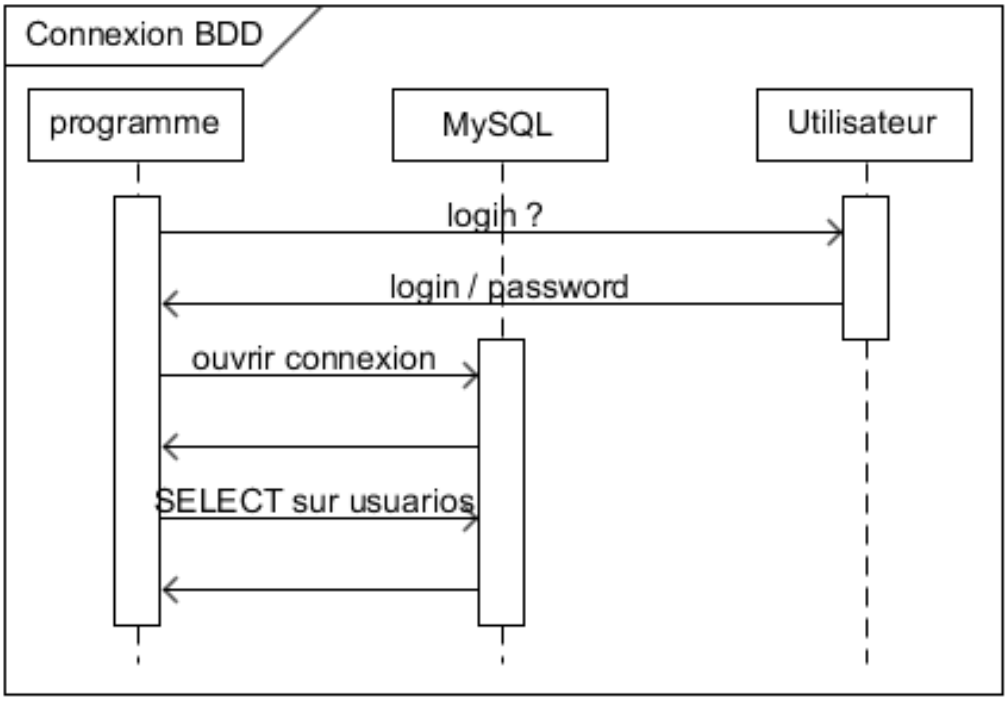


Figure 7 : Architecture particulière de l'application

[18] CERAMI, E., "Web services", O'Reilly & Associates, United States of America, 2002, Pages 1-34.

11. L'exécutable

Une fois la base de données créée, sécurisée et mise en place, un dernier problème doit être réglé. Comment déployer l'application Python facilement auprès de l'ensemble du personnel médical des divers laboratoires et centres de santé n'ayant pratiquement aucune notion d'informatique?

Il était inenvisageable d'installer sur chaque ordinateur tous les programmes nécessaires (Python 3.4 et PyQt5). Il aurait été possible de former quelques personnes à l'installation mais une fois encore, il ne s'agirait pas de la solution optimale. Afin de s'en approcher un maximum, le choix s'est arrêté sur la création d'un exécutable. Ce dernier est un programme standalone, c'est-à-dire qu'il contient lui-même tous les composants dont il a besoin pour fonctionner correctement. Un tel fichier à l'avantage de pouvoir installer tous les logiciels et programmes, comme ici Python, nécessaires au bon fonctionnement de l'application. Il crée également un lien à utiliser une fois le fichier décompressé. En effet, il suffit de lancer le fichier « codepo.exe » pour lancer l'application. De cette façon, il suffira de mettre le fichier exécutable en ligne ou de faire circuler des clés USB contenant ce même exécutable et tout le personnel pourra lancer l'application en un clic.

Dans un premier temps, le logiciel py2exe a été utilisé pour créer l'exécutable. Bien que conseillé sur de nombreux sites, il s'est avéré, par la suite, que l'exécutable généré comportait sporadiquement des erreurs. En effet, il était perçu comme un virus par certains systèmes d'exploitation. De plus, cet exécutable n'était compatible qu'avec le système d'exploitation Windows et des demandes ont été faites pour des ordinateurs de type MacOS.[19]

Un état de l'art a donc été réalisé et le choix s'est porté sur cx_freeze. Ce choix s'est avéré relativement évident, puisque ce logiciel générateur d'exécutables se trouve parmi les plus répandus sur le marché, est réputé facile à utiliser et surtout, a l'avantage de fonctionner sur plusieurs plateformes, dont Windows, Linux et MacOS.[20]

Le fichier setup.py qui permet de générer l'exécutable a été placé en annexe (cf. Annexe C).

[19] PY2EXE, http://www.py2exe.org/, <en ligne >, consulté le 8/02/2015.
[20] OPENCLASSROOMS, *Distribuer facilement nos programmes Python avec cx_Freeze,* http://openclassrooms.com/courses/apprenez-a-programmer-en-python/distribuer-facilement-nos-programmes-python-avec-cx-freeze, <en ligne>, consulté le 13/02/2015.
SOURCEFORGE, *cx_Freeze*, http://cx-freeze.sourceforge.net/, <en ligne>, consulté le 13/02/2015.

12. Conclusion

Ce rapport touchant à sa fin, nous tâcherons ici d'en rappeler les points importants, d'analyser le travail accompli par rapport au cahier des charges et de considérer le trajet parcouru d'un point de vue objectif.

En ce qui concerne la dynamique de groupe et les relations avec nos partenaires, la première étape du projet a été la prise de contact avec les acteurs belges et boliviens. En Belgique, la communication a été relativement rapide et aisée, tandis que le contact avec les Boliviens a mis plus de temps à s'établir. Une fois la relation mise en place avec le Sud, certains nous ont fait part de leur engouement, ainsi que de leur détermination à faire diminuer de manière significative le taux de mortalité lié au cancer du col de l'utérus. A la suite de ces différents échanges, un cahier des charges a vu le jour sur base d'un commun accord. Ce n'est qu'après cela que l'aspect technique a pu réellement démarrer. Par rapport au fonctionnement interne du groupe, il a également fallu prendre ses marques. Nous avons été mis face à nous-mêmes et aux autres afin de faire avancer ce projet. En plus de l'organisation et la discipline, cela est passé par le développement de compétences telles que l'argumentation de ses opinions et l'écoute des autres.

Au niveau technique, avant de se lancer dans la réalisation pratique de la base de données, de nouveaux outils et notions ont dû être maitrisées : MySQL, PyQt, notions de serveur, etc. Grâce à ces connaissances et notre travail, un système d'échange de données informatisé a pu voir le jour. Il se base sur le système de gestion de base de données MySQL 5.6, tandis que l'application est codée à l'aide du programme Python 3.4, et les interfaces reposent sur la bibliothèque PyQt5. Le tout est centralisé sur un serveur dédié OVH basé en France. Le format modèle-vue-contrôleur et la structure redondante du code ont été utilisés dans le but de faciliter la maintenance du programme par des informaticiens ou d'autres étudiants. Le programme a été pensé afin de répondre aux différentes attentes des locaux reprises dans le cahier des charges. Ce dernier a été rempli dans sa totalité mais a évolué tout au long du projet suite aux différents retours émis par nos partenaires en Bolivie. De cette collaboration, des applications et fonctionnalités supplémentaires ont vu le jour. Parmi ces optimisations, la principale offre la possibilité de supprimer, modifier ou ajouter des tests afin d'assurer la pérénité du projet.

A présent, le projet n'est terminé que dans sa partie théorique et purement académique mais il n'a pas encore débuté dans son application pratique. Comment le déploiement du logiciel se déroulera-t-il sur place ? Qu'en sera-t-il de la formation des médecins ? A quel point le personnel médical utilisera-t-il la base de données et sera satisfait du rendu? Quelles mises à jour devront être réalisées sur place après discussion avec les locaux ? Tant d'interrogations et de questionnements encore non résolus auxquels nous devront répondre dans notre mission d'ingénieur. A la fin de notre séjour, sur base de notre expérience de terrain, un document qui reprendra les différentes améliorations et fonctionnalités

nécessaires sera rédigé afin de guider au mieux de potentiels futurs développeurs de la base de données.

En guise de conclusion, rappelons le but de ce projet : faire baisser le taux de mortalité lié au cancer du col de l'utérus en améliorant les communications dans la province de Cochabamba. L'accomplissement de cet objectif ne pourra se vérifier que dans la durée, mais la base de données est fonctionnelle et intuitive et cela devrait se passer sans encombres. Il ne reste maintenant plus qu'à espérer que l'expérience de terrain en juillet 2015 soit au-delà de nos espérances d'un point de vue professionnel et surtout humain.

Sources

Sitographie

- DEVELOPPEZ.COM, *Le SQL de A à Z : 1ère partie – bases de données, SQL et types de données*, http://sqlpro.developpez.com/cours/sqlaz/fondements/#L5.1 , <en ligne>, consulté le 19/10/2014.

- DIDIER, J.- Y., *Introduction au réseau,* http://lsc.univ-evry.fr/~didier/webpage/pedagogie/ii25_final.pdf, <en ligne>, consulté le 14/02/15.

- EDOCEO, *Comma Separated Values (CSV) Standard File Format*, http://edoceo.com/utilitas/csv-file-format, <en ligne>, consulté le 5/11/2014.

- LE JOURNAL DU NET, *Comparatif : 14 langages au crible*, http://www.journaldunet.com/developpeur/tutoriel/out/040728-comparaison-langages1b.shtml, <en ligne>, consulté le 20/10/2014.

- MONO, *GtkSharp*, http://www.mono-project.com/docs/gui/gtksharp/, <en ligne >, consulté le 21/10/2014.

- MYSQL, *MySQL 5.6 Reference Manual*, http://dev.mysql.com/doc/refman/5.6/en/index.html, <en ligne>, consulté le 19/10/2014.

- NEXTADVISOR, *Dedicated vs. Shared Hosting: What's the difference?,* http://www.nextadvisor.com/blog/2014/02/06/dedicated-vs-shared-hosting-difference/, <en ligne>, consulté le 18/02/2015.

- OPENCLASSROOMS, *Distribuer facilement nos programmes Python avec cx_Freeze,* http://openclassrooms.com/courses/apprenez-a-programmer-en-python/distribuer-facilement-nos-programmes-python-avec-cx-freeze, <en ligne>, consulté le 13/02/2015

- PAVLIDIS, T., *Fundamentals of X Programming: Graphical User Interfaces and Beyond,* New York, Kluwer Academic Publishers, 2002, eBook: http://imcs.dvfu.ru/lib.int/docs/Programming/Kluwer%20Academic%20Publishers%20-%20Fundamentals%20of%20X%20Programming%20GUI%20and%20Beyond.pdf , consulté le 19/10/2014.

- PUISEUX, P., *Tutoriel sur PyQt : Présentation,* http://web.univ-pau.fr/~puiseux/enseignement/python/tuto-PyQt.01%28presentation%29.pdf, <en ligne>, consulté le 20/10/2014.

- PYTHON, *MySQL*, https://wiki.python.org/moin/MySQL, <en ligne>, consulté le 21/10/2014.

- PYTHON, *Tkinter – Python interface to Tcl/Tk*, https://docs.python.org/2/library/tkinter.html, <en ligne >, consulté le 21/10/2014.

- PY2EXE, http://www.py2exe.org/, <en ligne >, consulté le 8/02/2015.

- QT, *Qt Documentation : Qt Widgets*, http://doc.qt.io/qt-5/qtwidgets-index.html , <en ligne>, consulté le 21/10/2014.

- SCENARI-PLATFORM, *BD et SGBD : vue d'ensemble,* http://scenari-platform.org/mobile-source/opale-demo/co/01_introduction_web/co/AC01vue.html , <en ligne>, consulté le 19/10/2014.

- SOURCEFORGE, *cx_Freeze*, http://cx-freeze.sourceforge.net/, <en ligne>, consulté le 13/02/2015.

- TAQUET, P. (Solidarité mondiale), *Santé en Bolivie : « pour les gens, l'avenir est une torture »*, http://www.solmond.be/pour-les-gens-l-avenir-est-une, <en ligne>, consulté le 19/02/2015.

- W3SCHOOLS.COM, *SQL,*http://www.w3schools.com/sql/sql_select.asp , <en ligne>, consulté le 18/10/2014.

- YUNES, J.-B., *Interfaces graphiques (GUI)*, http://www.liafa.univ-paris-diderot.fr/~carton/Enseignement/InterfacesGraphiques/MasterInfo/Cours/Swing/mvc.html, <en ligne>, consulté le 14/02/15.

Bibliographie

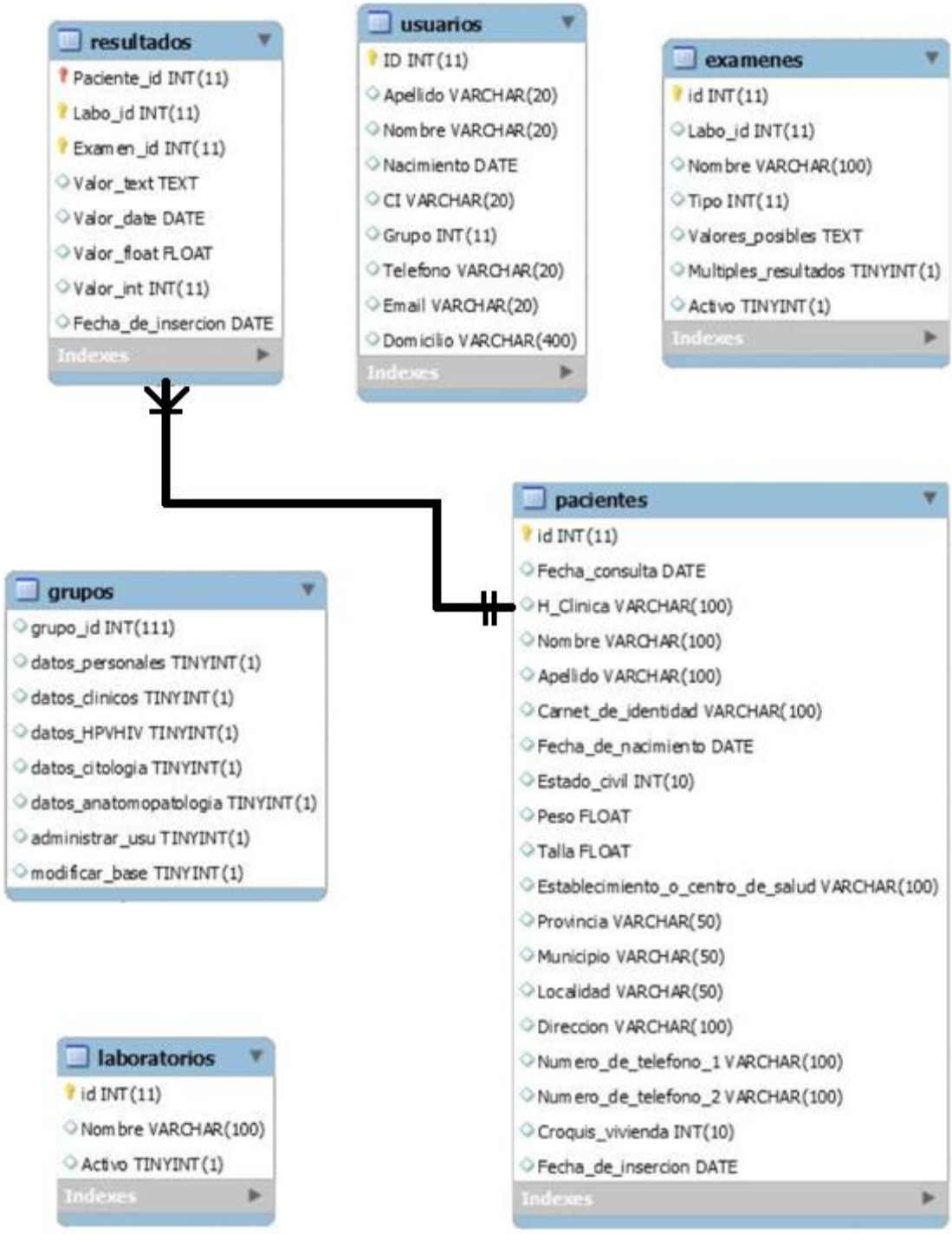
- CERAMI, E., "Web services", O'Reilly & Associates, United States of America, 2002.
- CIUBOTARU, B., et al. "Advanced Network Programming – Principles and Techniques", Springer, Londres, 2013.
- POPULATION REFERENCE BUREAU (PRB), ALLIANCE FOR CERVICAL CANCER PREVENTION (ACCP), "Prévenir le cancer du col de l'utérus de par le monde", 2004.
- SUMMERFIELD, M., "Rapid GUI Programming with Python and Qt - the definitive guide to PyQt programming", Prentice Hall, 2007.

Annexes

Annexe A: Code d'import de données Excel vers une base de données

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

""" This file represents a model for the transfert of the data of an excel
file to a MySQL database
"""

from PyQt5.QtCore import *
from PyQt5.QtSql import *
from PyQt5.QtWidgets import *


app = QApplication([])

# Connect to the database
username = input("Nom d'utilisateur base de donnée : ")
password = input("Mot de passe : ")
dbname = input("Nom de la base de donnée : ")

db = QSqlDatabase("QMYSQL")
db.setHostName("localhost")
db.setUserName(username)
db.setPassword(password)
db.setDatabaseName(dbname)

if not db.open():
    print("Impossible de se connecter à la base de donnée : %s" % db.lastError().databaseText())
    exit()

query = QSqlQuery(db)


# Global version

# With an input(), the '\' are changed to '\\'
file = input("Chemin du fichier excell enregistré sous csv:")
fichier = open(file,"r")

# List of lists
colonnes=[]
    for line in fichier:
    parts = line.strip().split(';')
    for i in range(len(parts)):
        if len(colonnes)<=i:
            colonnes.append([])
        colonnes[i].append(parts[i])
query.prepare("INSERT INTO personnes(nom,prenom) VALUES (" + ','.join('?' for colonne in colonnes) + ");")

for colonne in colonnes:
    query.addBindValue(colonne)

# query.execBatch() is called once for all the querys and it is faster
# than an iteration with for and exec_()

query.execBatch()
```

Annexe B: Tables de la base de données « codepo »



Annexe C : Application Python

Ui fichiers (vue)

logindialog.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'logindialog.ui'
#
# Created by: PyQt5 UI code generator 5.3.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_LoginDialog(object):
    def setupUi(self, LoginDialog):
        LoginDialog.setObjectName("LoginDialog")
        LoginDialog.resize(283, 122)
        self.verticalLayout = QtWidgets.QVBoxLayout(LoginDialog)
        self.verticalLayout.setObjectName("verticalLayout")
        self.formLayout = QtWidgets.QFormLayout()
        self.formLayout.setObjectName("formLayout")
        self.lblEmail = QtWidgets.QLabel(LoginDialog)
        self.lblEmail.setObjectName("lblEmail")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.lblEmail)
        self.txtEmail = QtWidgets.QLineEdit(LoginDialog)
        self.txtEmail.setObjectName("txtEmail")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.txtEmail)
        self.lblPassword = QtWidgets.QLabel(LoginDialog)
        self.lblPassword.setObjectName("lblPassword")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lblPassword)
        self.txtPassword = QtWidgets.QLineEdit(LoginDialog)
        self.txtPassword.setEchoMode(QtWidgets.QLineEdit.Password)
        self.txtPassword.setObjectName("txtPassword")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.txtPassword)
        self.verticalLayout.addLayout(self.formLayout)
        spacerItem = QtWidgets.QSpacerItem(20, 28, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem)
        self.buttons = QtWidgets.QDialogButtonBox(LoginDialog)
        self.buttons.setOrientation(QtCore.Qt.Horizontal)
        self.buttons.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Ok)
        self.buttons.setObjectName("buttons")
        self.verticalLayout.addWidget(self.buttons)

        self.retranslateUi(LoginDialog)
        self.buttons.accepted.connect(LoginDialog.accept)
        self.buttons.rejected.connect(LoginDialog.reject)
        QtCore.QMetaObject.connectSlotsByName(LoginDialog)

    def retranslateUi(self, LoginDialog):
        _translate = QtCore.QCoreApplication.translate
        LoginDialog.setWindowTitle(_translate("LoginDialog", "Login"))
        self.lblEmail.setText(_translate("LoginDialog", "Cédula de identidad:"))
        self.lblPassword.setText(_translate("LoginDialog", "Contraseña:"))
```

ui_mainwindow.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'mainwindow.ui'
#
# Created: Thu Apr  2 10:27:05 2015
#      by: PyQt5 UI code generator 5.2.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(680, 505)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_4.setSpacing(0)
        self.horizontalLayout_4.setContentsMargins(-1, 10, -1, -1)
        self.horizontalLayout_4.setObjectName("horizontalLayout_4")
        spacerItem = QtWidgets.QSpacerItem(60, 20, QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_4.addItem(spacerItem)
        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setObjectName("pushButton_2")
        self.horizontalLayout_4.addWidget(self.pushButton_2)
        spacerItem1 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_4.addItem(spacerItem1)
        self.btnDesconectarses = QtWidgets.QPushButton(self.centralwidget)
        self.btnDesconectarses.setObjectName("btnDesconectarses")
        self.horizontalLayout_4.addWidget(self.btnDesconectarses)
        spacerItem2 = QtWidgets.QSpacerItem(60, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_4.addItem(spacerItem2)
        self.verticalLayout_2.addLayout(self.horizontalLayout_4)
        self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)
        self.tabWidget.setObjectName("tabWidget")
        self.tabUserList = QtWidgets.QWidget()
        self.tabUserList.setObjectName("tabUserList")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.tabUserList)
        self.verticalLayout.setObjectName("verticalLayout")
        self.userList = UserList(self.tabUserList)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.userList.sizePolicy().hasHeightForWidth())
        self.userList.setSizePolicy(sizePolicy)
        self.userList.setObjectName("userList")
        self.verticalLayout.addWidget(self.userList)
        self.tabWidget.addTab(self.tabUserList, "")
        self.tabPatientList = QtWidgets.QWidget()
        self.tabPatientList.setObjectName("tabPatientList")
        self.verticalLayout_3 = QtWidgets.QVBoxLayout(self.tabPatientList)
        self.verticalLayout_3.setObjectName("verticalLayout_3")
        self.patientList = PatientList(self.tabPatientList)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.patientList.sizePolicy().hasHeightForWidth())
        self.patientList.setSizePolicy(sizePolicy)
```

```
        self.patientList.setObjectName("patientList")
        self.verticalLayout_3.addWidget(self.patientList)
        self.tabWidget.addTab(self.tabPatientList, "")
        self.tabLaboList = QtWidgets.QWidget()
        self.tabLaboList.setObjectName("tabLaboList")
        self.verticalLayout_4 = QtWidgets.QVBoxLayout(self.tabLaboList)
        self.verticalLayout_4.setObjectName("verticalLayout_4")
        self.laboList = LaboList(self.tabLaboList)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.laboList.sizePolicy().hasHeightForWidth())
        self.laboList.setSizePolicy(sizePolicy)
        self.laboList.setObjectName("laboList")
        self.verticalLayout_4.addWidget(self.laboList)
        self.tabWidget.addTab(self.tabLaboList, "")
        self.tabTestList = QtWidgets.QWidget()
        self.tabTestList.setObjectName("tabTestList")
        self.verticalLayout_5 = QtWidgets.QVBoxLayout(self.tabTestList)
        self.verticalLayout_5.setObjectName("verticalLayout_5")
        self.testList = TestList(self.tabTestList)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.testList.sizePolicy().hasHeightForWidth())
        self.testList.setSizePolicy(sizePolicy)
        self.testList.setObjectName("testList")
        self.verticalLayout_5.addWidget(self.testList)
        self.tabWidget.addTab(self.tabTestList, "")
        self.tabExcel = QtWidgets.QWidget()
        self.tabExcel.setObjectName("tabExcel")
        self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.tabExcel)
        self.verticalLayout_6.setObjectName("verticalLayout_6")
        spacerItem3 = QtWidgets.QSpacerItem(20, 98, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout_6.addItem(spacerItem3)
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setContentsMargins(-1, 20, -1, -1)
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.label = QtWidgets.QLabel(self.tabExcel)
        self.label.setObjectName("label")
        self.horizontalLayout_2.addWidget(self.label)
        self.btnVeronique = QtWidgets.QPushButton(self.tabExcel)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.btnVeronique.sizePolicy().hasHeightForWidth())
        self.btnVeronique.setSizePolicy(sizePolicy)
        self.btnVeronique.setMinimumSize(QtCore.QSize(0, 0))
        self.btnVeronique.setObjectName("btnVeronique")
        self.horizontalLayout_2.addWidget(self.btnVeronique)
        self.verticalLayout_6.addLayout(self.horizontalLayout_2)
        spacerItem4 = QtWidgets.QSpacerItem(20, 99, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout_6.addItem(spacerItem4)
        self.label_2 = QtWidgets.QLabel(self.tabExcel)
        self.label_2.setObjectName("label_2")
        self.verticalLayout_6.addWidget(self.label_2)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.lblDel = QtWidgets.QLabel(self.tabExcel)
        self.lblDel.setObjectName("lblDel")
        self.horizontalLayout.addWidget(self.lblDel)
        self.dtDel = QtWidgets.QDateEdit(self.tabExcel)
        self.dtDel.setObjectName("dtDel")
        self.horizontalLayout.addWidget(self.dtDel)
        self.lblAl = QtWidgets.QLabel(self.tabExcel)
        self.lblAl.setObjectName("lblAl")
```

```
        self.horizontalLayout.addWidget(self.lblAl)
        self.dtAl = QtWidgets.QDateEdit(self.tabExcel)
        self.dtAl.setObjectName("dtAl")
        self.horizontalLayout.addWidget(self.dtAl)
        self.btnExcel = QtWidgets.QPushButton(self.tabExcel)
        self.btnExcel.setMinimumSize(QtCore.QSize(0, 0))
        self.btnExcel.setObjectName("btnExcel")
        self.horizontalLayout.addWidget(self.btnExcel)
        self.verticalLayout_6.addLayout(self.horizontalLayout)
        spacerItem5 = QtWidgets.QSpacerItem(20, 150, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout_6.addItem(spacerItem5)
        self.tabWidget.addTab(self.tabExcel, "")
        self.verticalLayout_2.addWidget(self.tabWidget)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 680, 25))
        self.menubar.setObjectName("menubar")
        self.mnuFile = QtWidgets.QMenu(self.menubar)
        self.mnuFile.setObjectName("mnuFile")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.mnuQuit = QtWidgets.QAction(MainWindow)
        self.mnuQuit.setObjectName("mnuQuit")
        self.mnuFile.addAction(self.mnuQuit)
        self.menubar.addAction(self.mnuFile.menuAction())

        self.retranslateUi(MainWindow)
        self.tabWidget.setCurrentIndex(4)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Menu principal"))
        self.pushButton_2.setText(_translate("MainWindow", "Completar datos personales"))
        self.btnDesconectarses.setText(_translate("MainWindow", "Desconectarse de la base de datos"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tabUserList), _translate("MainWindow", "Usuarios"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tabPatientList), _translate("MainWindow",
"Pacientes"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tabLaboList), _translate("MainWindow",
"Laboratorios"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tabTestList), _translate("MainWindow", "Entradas"))
        self.label.setText(_translate("MainWindow", "Generar la tabla con los tests especificados para los estudios
estadisticos"))
        self.btnVeronique.setText(_translate("MainWindow", "Estadisticas"))
        self.label_2.setText(_translate("MainWindow", "Buscar los nuevos pacientes añadidos o modificados en el
rango de fechas:"))
        self.lblDel.setText(_translate("MainWindow", "Inicio:"))
        self.lblAl.setText(_translate("MainWindow", "Fin:"))
        self.btnExcel.setText(_translate("MainWindow", "Lista Pacientes"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tabExcel), _translate("MainWindow", "Excel"))
        self.mnuFile.setTitle(_translate("MainWindow", "&File"))
        self.mnuQuit.setText(_translate("MainWindow", "&Quit..."))

from userlist import UserList
from testlist import TestList
from labolist import LaboList
from patientlist import PatientList
```

ui_userdialog.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'userdialog.ui'
#
# Created: Tue Mar 31 16:07:14 2015
#      by: PyQt5 UI code generator 5.2.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_UserDialog(object):
    def setupUi(self, UserDialog):
        UserDialog.setObjectName("UserDialog")
        UserDialog.resize(430, 532)
        self.verticalLayout = QtWidgets.QVBoxLayout(UserDialog)
        self.verticalLayout.setObjectName("verticalLayout")
        spacerItem = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem)
        self.formLayout = QtWidgets.QFormLayout()
        self.formLayout.setFieldGrowthPolicy(QtWidgets.QFormLayout.ExpandingFieldsGrow)
        self.formLayout.setObjectName("formLayout")
        self.lblFirstName = QtWidgets.QLabel(UserDialog)
        self.lblFirstName.setObjectName("lblFirstName")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.lblFirstName)
        self.txtFirstName = QtWidgets.QLineEdit(UserDialog)
        self.txtFirstName.setObjectName("txtFirstName")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.txtFirstName)
        self.lblLastName = QtWidgets.QLabel(UserDialog)
        self.lblLastName.setObjectName("lblLastName")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lblLastName)
        self.txtLastName = QtWidgets.QLineEdit(UserDialog)
        self.txtLastName.setObjectName("txtLastName")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.txtLastName)
        self.verticalLayout.addLayout(self.formLayout)
        spacerItem1 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem1)
        self.line = QtWidgets.QFrame(UserDialog)
        self.line.setFrameShape(QtWidgets.QFrame.HLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.verticalLayout.addWidget(self.line)
        spacerItem2 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem2)
        self.formLayout_2 = QtWidgets.QFormLayout()
        self.formLayout_2.setFieldGrowthPolicy(QtWidgets.QFormLayout.AllNonFixedFieldsGrow)
        self.formLayout_2.setObjectName("formLayout_2")
        self.lblCI = QtWidgets.QLabel(UserDialog)
        self.lblCI.setObjectName("lblCI")
        self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.lblCI)
        self.txtCI = QtWidgets.QLineEdit(UserDialog)
        self.txtCI.setObjectName("txtCI")
        self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.txtCI)
        self.lblPassword = QtWidgets.QLabel(UserDialog)
        self.lblPassword.setObjectName("lblPassword")
        self.formLayout_2.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lblPassword)
        self.txtPassword = QtWidgets.QLineEdit(UserDialog)
        self.txtPassword.setEchoMode(QtWidgets.QLineEdit.Normal)
        self.txtPassword.setObjectName("txtPassword")
        self.formLayout_2.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.txtPassword)
        self.verticalLayout.addLayout(self.formLayout_2)
```

```
        self.labelError1 = QtWidgets.QLabel(UserDialog)
        self.labelError1.setText("")
        self.labelError1.setObjectName("labelError1")
        self.verticalLayout.addWidget(self.labelError1)
        spacerItem3 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem3)
        self.crearpass = QtWidgets.QPushButton(UserDialog)
        self.crearpass.setObjectName("crearpass")
        self.verticalLayout.addWidget(self.crearpass)
        self.line_2 = QtWidgets.QFrame(UserDialog)
        self.line_2.setFrameShape(QtWidgets.QFrame.HLine)
        self.line_2.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line_2.setObjectName("line_2")
        self.verticalLayout.addWidget(self.line_2)
        self.label = QtWidgets.QLabel(UserDialog)
        self.label.setObjectName("label")
        self.verticalLayout.addWidget(self.label)
        self.radioButton = QtWidgets.QRadioButton(UserDialog)
        self.radioButton.setObjectName("radioButton")
        self.verticalLayout.addWidget(self.radioButton)
        self.radioButton_2 = QtWidgets.QRadioButton(UserDialog)
        self.radioButton_2.setObjectName("radioButton_2")
        self.verticalLayout.addWidget(self.radioButton_2)
        self.radioButton_3 = QtWidgets.QRadioButton(UserDialog)
        self.radioButton_3.setObjectName("radioButton_3")
        self.verticalLayout.addWidget(self.radioButton_3)
        self.radioButton_4 = QtWidgets.QRadioButton(UserDialog)
        self.radioButton_4.setObjectName("radioButton_4")
        self.verticalLayout.addWidget(self.radioButton_4)
        spacerItem4 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem4)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setSizeConstraint(QtWidgets.QLayout.SetNoConstraint)
        self.horizontalLayout.setObjectName("horizontalLayout")
        spacerItem5 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout.addItem(spacerItem5)
        self.verticalLayout_2 = QtWidgets.QVBoxLayout()
        self.verticalLayout_2.setSizeConstraint(QtWidgets.QLayout.SetNoConstraint)
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.lblIndico = QtWidgets.QLabel(UserDialog)
        self.lblIndico.setObjectName("lblIndico")
        self.verticalLayout_2.addWidget(self.lblIndico)
        self.checkBox_Modif = QtWidgets.QCheckBox(UserDialog)
        self.checkBox_Modif.setObjectName("checkBox_Modif")
        self.verticalLayout_2.addWidget(self.checkBox_Modif)
        self.checkBox_gestio = QtWidgets.QCheckBox(UserDialog)
        self.checkBox_gestio.setObjectName("checkBox_gestio")
        self.verticalLayout_2.addWidget(self.checkBox_gestio)
        self.horizontalLayout.addLayout(self.verticalLayout_2)
        spacerItem6 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout.addItem(spacerItem6)
        self.verticalLayout.addLayout(self.horizontalLayout)
        spacerItem7 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem7)
        self.labelError2 = QtWidgets.QLabel(UserDialog)
        self.labelError2.setText("")
        self.labelError2.setObjectName("labelError2")
        self.verticalLayout.addWidget(self.labelError2)
        self.buttons = QtWidgets.QDialogButtonBox(UserDialog)
        self.buttons.setOrientation(QtCore.Qt.Horizontal)
        self.buttons.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Save)
        self.buttons.setObjectName("buttons")
        self.verticalLayout.addWidget(self.buttons)
```

```
        self.retranslateUi(UserDialog)
        self.buttons.rejected.connect(UserDialog.reject)
        QtCore.QMetaObject.connectSlotsByName(UserDialog)

    def retranslateUi(self, UserDialog):
        _translate = QtCore.QCoreApplication.translate
        UserDialog.setWindowTitle(_translate("UserDialog", "Añadir o modificar un usuario"))
        self.lblFirstName.setText(_translate("UserDialog", "Nombre:"))
        self.lblLastName.setText(_translate("UserDialog", "Apellido:"))
        self.lblCI.setText(_translate("UserDialog", "Cédula de identidad:"))
        self.lblPassword.setText(_translate("UserDialog", "Contraseña:"))
        self.crearpass.setText(_translate("UserDialog", "Crear contraseña aleatoria"))
        self.label.setText(_translate("UserDialog", "Indique la categoría del nuevo usuario:"))
        self.radioButton.setText(_translate("UserDialog", "Personal de sanidad"))
        self.radioButton_2.setText(_translate("UserDialog", "Laboratorio de HPV-HIV"))
        self.radioButton_3.setText(_translate("UserDialog", "Laboratorio de citología"))
        self.radioButton_4.setText(_translate("UserDialog", "Laboratorio de anatomopatología"))
        self.lblIndico.setText(_translate("UserDialog", "Indique los privilegios:"))
        self.checkBox_Modif.setText(_translate("UserDialog", "Modificar la base de datos"))
        self.checkBox_gestio.setText(_translate("UserDialog", "Gestionar los usuarios"))
```

ui_personaldialog.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'personaldialog.ui'
#
# Created by: PyQt5 UI code generator 5.4.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_PersonalDialog(object):
    def setupUi(self, PersonalDialog):
        PersonalDialog.setObjectName("PersonalDialog")
        PersonalDialog.resize(438, 348)
        self.verticalLayout = QtWidgets.QVBoxLayout(PersonalDialog)
        self.verticalLayout.setObjectName("verticalLayout")
        self.formLayout = QtWidgets.QFormLayout()
        self.formLayout.setObjectName("formLayout")
        self.lblNombre = QtWidgets.QLabel(PersonalDialog)
        self.lblNombre.setObjectName("lblNombre")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.lblNombre)
        self.txtNombre = QtWidgets.QLineEdit(PersonalDialog)
        self.txtNombre.setObjectName("txtNombre")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.txtNombre)
        self.lblApellido = QtWidgets.QLabel(PersonalDialog)
        self.lblApellido.setObjectName("lblApellido")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lblApellido)
        self.txtApellido = QtWidgets.QLineEdit(PersonalDialog)
        self.txtApellido.setEchoMode(QtWidgets.QLineEdit.Normal)
        self.txtApellido.setObjectName("txtApellido")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.txtApellido)
        self.lblCI = QtWidgets.QLabel(PersonalDialog)
        self.lblCI.setObjectName("lblCI")
        self.formLayout.setWidget(2, QtWidgets.QFormLayout.LabelRole, self.lblCI)
        self.lblTelef = QtWidgets.QLabel(PersonalDialog)
        self.lblTelef.setObjectName("lblTelef")
        self.formLayout.setWidget(4, QtWidgets.QFormLayout.LabelRole, self.lblTelef)
        self.lblEmail = QtWidgets.QLabel(PersonalDialog)
        self.lblEmail.setObjectName("lblEmail")
        self.formLayout.setWidget(5, QtWidgets.QFormLayout.LabelRole, self.lblEmail)
        self.lblDomicilio = QtWidgets.QLabel(PersonalDialog)
        self.lblDomicilio.setObjectName("lblDomicilio")
```

```
        self.formLayout.setWidget(6, QtWidgets.QFormLayout.LabelRole, self.lblDomicilio)
        self.txtCI = QtWidgets.QLineEdit(PersonalDialog)
        self.txtCI.setObjectName("txtCI")
        self.formLayout.setWidget(2, QtWidgets.QFormLayout.FieldRole, self.txtCI)
        self.txtTelef = QtWidgets.QLineEdit(PersonalDialog)
        self.txtTelef.setObjectName("txtTelef")
        self.formLayout.setWidget(4, QtWidgets.QFormLayout.FieldRole, self.txtTelef)
        self.txtEmail = QtWidgets.QLineEdit(PersonalDialog)
        self.txtEmail.setObjectName("txtEmail")
        self.formLayout.setWidget(5, QtWidgets.QFormLayout.FieldRole, self.txtEmail)
        self.txtDomicilio = QtWidgets.QLineEdit(PersonalDialog)
        self.txtDomicilio.setObjectName("txtDomicilio")
        self.formLayout.setWidget(6, QtWidgets.QFormLayout.FieldRole, self.txtDomicilio)
        self.lblNacimiento = QtWidgets.QLabel(PersonalDialog)
        self.lblNacimiento.setObjectName("lblNacimiento")
        self.formLayout.setWidget(3, QtWidgets.QFormLayout.LabelRole, self.lblNacimiento)
        self.dateEdit = QtWidgets.QDateEdit(PersonalDialog)
        self.dateEdit.setObjectName("dateEdit")
        self.formLayout.setWidget(3, QtWidgets.QFormLayout.FieldRole, self.dateEdit)
        self.verticalLayout.addLayout(self.formLayout)
        spacerItem = QtWidgets.QSpacerItem(20, 28, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Minimum)
        self.verticalLayout.addItem(spacerItem)
        self.formLayout_2 = QtWidgets.QFormLayout()
        self.formLayout_2.setObjectName("formLayout_2")
        self.lblPassword = QtWidgets.QLabel(PersonalDialog)
        self.lblPassword.setObjectName("lblPassword")
        self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.lblPassword)
        self.txtPassword = QtWidgets.QLineEdit(PersonalDialog)
        self.txtPassword.setObjectName("txtPassword")
        self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.txtPassword)
        self.lblPassword2 = QtWidgets.QLabel(PersonalDialog)
        self.lblPassword2.setObjectName("lblPassword2")
        self.formLayout_2.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lblPassword2)
        self.txtPassword2 = QtWidgets.QLineEdit(PersonalDialog)
        self.txtPassword2.setObjectName("txtPassword2")
        self.formLayout_2.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.txtPassword2)
        self.verticalLayout.addLayout(self.formLayout_2)
        spacerItem1 = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Minimum)
        self.verticalLayout.addItem(spacerItem1)
        self.lblError = QtWidgets.QLabel(PersonalDialog)
        palette = QtGui.QPalette()
        brush = QtGui.QBrush(QtGui.QColor(255, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.WindowText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Text, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.WindowText, brush)
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Text, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.WindowText, brush)
        brush = QtGui.QBrush(QtGui.QColor(120, 120, 120))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Text, brush)
        self.lblError.setPalette(palette)
        self.lblError.setText("")
        self.lblError.setObjectName("lblError")
        self.verticalLayout.addWidget(self.lblError)
        self.buttons = QtWidgets.QDialogButtonBox(PersonalDialog)
        self.buttons.setOrientation(QtCore.Qt.Horizontal)
        self.buttons.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Save)
```

```
        self.buttons.setObjectName("buttons")
        self.verticalLayout.addWidget(self.buttons)

        self.retranslateUi(PersonalDialog)
        self.buttons.rejected.connect(PersonalDialog.reject)
        QtCore.QMetaObject.connectSlotsByName(PersonalDialog)
        PersonalDialog.setTabOrder(self.txtNombre, self.txtApellido)
        PersonalDialog.setTabOrder(self.txtApellido, self.txtCI)
        PersonalDialog.setTabOrder(self.txtCI, self.dateEdit)
        PersonalDialog.setTabOrder(self.dateEdit, self.txtTelef)
        PersonalDialog.setTabOrder(self.txtTelef, self.txtEmail)
        PersonalDialog.setTabOrder(self.txtEmail, self.txtDomicilio)
        PersonalDialog.setTabOrder(self.txtDomicilio, self.txtPassword)
        PersonalDialog.setTabOrder(self.txtPassword, self.txtPassword2)

    def retranslateUi(self, PersonalDialog):
        _translate = QtCore.QCoreApplication.translate
        PersonalDialog.setWindowTitle(_translate("PersonalDialog", "Añadir o modificar los datos personales"))
        self.lblNombre.setText(_translate("PersonalDialog", "Nombre:"))
        self.lblApellido.setText(_translate("PersonalDialog", "Apellidos:"))
        self.lblCI.setText(_translate("PersonalDialog", "Cédula de identidad:"))
        self.lblTelef.setText(_translate("PersonalDialog", "Teléfono:"))
        self.lblEmail.setText(_translate("PersonalDialog", "Email:"))
        self.lblDomicilio.setText(_translate("PersonalDialog", "Domicilio:"))
        self.lblNacimiento.setText(_translate("PersonalDialog", "Fecha de nacimiento:"))
        self.dateEdit.setDisplayFormat(_translate("PersonalDialog", "dd/MM/yyyy"))
        self.lblPassword.setText(_translate("PersonalDialog", "Cambiar contraseña:"))
        self.lblPassword2.setText(_translate("PersonalDialog", "Confirmar contraseña:"))
```

ui_patientdialog.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'patientdialog.ui'
#
# Created: Sun Mar 29 09:40:42 2015
#      by: PyQt5 UI code generator 5.2.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_PatientDialog(object):
    def setupUi(self, PatientDialog):
        PatientDialog.setObjectName("PatientDialog")
        PatientDialog.resize(487, 631)
        self.verticalLayout = QtWidgets.QVBoxLayout(PatientDialog)
        self.verticalLayout.setObjectName("verticalLayout")
        self.formLayout = QtWidgets.QFormLayout()
        self.formLayout.setFieldGrowthPolicy(QtWidgets.QFormLayout.AllNonFixedFieldsGrow)
        self.formLayout.setObjectName("formLayout")
        self.lblFechaConsulta = QtWidgets.QLabel(PatientDialog)
        self.lblFechaConsulta.setObjectName("lblFechaConsulta")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.lblFechaConsulta)
        self.dtFechaConsulta = QtWidgets.QDateEdit(PatientDialog)
        self.dtFechaConsulta.setObjectName("dtFechaConsulta")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.dtFechaConsulta)
        self.lblHClinica = QtWidgets.QLabel(PatientDialog)
        self.lblHClinica.setObjectName("lblHClinica")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lblHClinica)
        self.txtHClinica = QtWidgets.QLineEdit(PatientDialog)
        self.txtHClinica.setObjectName("txtHClinica")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.txtHClinica)
        self.lblNombre = QtWidgets.QLabel(PatientDialog)
        self.lblNombre.setObjectName("lblNombre")
        self.formLayout.setWidget(2, QtWidgets.QFormLayout.LabelRole, self.lblNombre)
```

```
self.txtNombre = QtWidgets.QLineEdit(PatientDialog)
self.txtNombre.setObjectName("txtNombre")
self.formLayout.setWidget(2, QtWidgets.QFormLayout.FieldRole, self.txtNombre)
self.lblApellido = QtWidgets.QLabel(PatientDialog)
self.lblApellido.setObjectName("lblApellido")
self.formLayout.setWidget(3, QtWidgets.QFormLayout.LabelRole, self.lblApellido)
self.txtApellido = QtWidgets.QLineEdit(PatientDialog)
self.txtApellido.setObjectName("txtApellido")
self.formLayout.setWidget(3, QtWidgets.QFormLayout.FieldRole, self.txtApellido)
self.lblFechaDeNacimiento = QtWidgets.QLabel(PatientDialog)
self.lblFechaDeNacimiento.setObjectName("lblFechaDeNacimiento")
self.formLayout.setWidget(5, QtWidgets.QFormLayout.LabelRole, self.lblFechaDeNacimiento)
self.dtFechaDeNacimiento = QtWidgets.QDateEdit(PatientDialog)
self.dtFechaDeNacimiento.setObjectName("dtFechaDeNacimiento")
self.formLayout.setWidget(5, QtWidgets.QFormLayout.FieldRole, self.dtFechaDeNacimiento)
self.lblEstadoCivil_2 = QtWidgets.QLabel(PatientDialog)
self.lblEstadoCivil_2.setObjectName("lblEstadoCivil_2")
self.formLayout.setWidget(6, QtWidgets.QFormLayout.LabelRole, self.lblEstadoCivil_2)
self.cbEstadoCivil = QtWidgets.QComboBox(PatientDialog)
self.cbEstadoCivil.setObjectName("cbEstadoCivil")
self.cbEstadoCivil.addItem("")
self.cbEstadoCivil.setItemText(0, "")
self.cbEstadoCivil.addItem("")
self.cbEstadoCivil.addItem("")
self.cbEstadoCivil.addItem("")
self.cbEstadoCivil.addItem("")
self.cbEstadoCivil.addItem("")
self.cbEstadoCivil.addItem("")
self.formLayout.setWidget(6, QtWidgets.QFormLayout.FieldRole, self.cbEstadoCivil)
self.lblPeso = QtWidgets.QLabel(PatientDialog)
self.lblPeso.setObjectName("lblPeso")
self.formLayout.setWidget(7, QtWidgets.QFormLayout.LabelRole, self.lblPeso)
self.lblTalla = QtWidgets.QLabel(PatientDialog)
self.lblTalla.setObjectName("lblTalla")
self.formLayout.setWidget(8, QtWidgets.QFormLayout.LabelRole, self.lblTalla)
self.lblEstablecimientoOCentroDeSalud = QtWidgets.QLabel(PatientDialog)
self.lblEstablecimientoOCentroDeSalud.setObjectName("lblEstablecimientoOCentroDeSalud")
self.formLayout.setWidget(9, QtWidgets.QFormLayout.LabelRole, self.lblEstablecimientoOCentroDeSalud)
self.txtEstablecimientoOCentroDeSalud = QtWidgets.QLineEdit(PatientDialog)
self.txtEstablecimientoOCentroDeSalud.setObjectName("txtEstablecimientoOCentroDeSalud")
self.formLayout.setWidget(9, QtWidgets.QFormLayout.FieldRole, self.txtEstablecimientoOCentroDeSalud)
self.lblProvincia = QtWidgets.QLabel(PatientDialog)
self.lblProvincia.setObjectName("lblProvincia")
self.formLayout.setWidget(10, QtWidgets.QFormLayout.LabelRole, self.lblProvincia)
self.txtProvincia = QtWidgets.QLineEdit(PatientDialog)
self.txtProvincia.setObjectName("txtProvincia")
self.formLayout.setWidget(10, QtWidgets.QFormLayout.FieldRole, self.txtProvincia)
self.lblMunicipio = QtWidgets.QLabel(PatientDialog)
self.lblMunicipio.setObjectName("lblMunicipio")
self.formLayout.setWidget(11, QtWidgets.QFormLayout.LabelRole, self.lblMunicipio)
self.txtMunicipio = QtWidgets.QLineEdit(PatientDialog)
self.txtMunicipio.setObjectName("txtMunicipio")
self.formLayout.setWidget(11, QtWidgets.QFormLayout.FieldRole, self.txtMunicipio)
self.lblLocalidad = QtWidgets.QLabel(PatientDialog)
self.lblLocalidad.setObjectName("lblLocalidad")
self.formLayout.setWidget(12, QtWidgets.QFormLayout.LabelRole, self.lblLocalidad)
self.txtLocalidad = QtWidgets.QLineEdit(PatientDialog)
self.txtLocalidad.setObjectName("txtLocalidad")
self.formLayout.setWidget(12, QtWidgets.QFormLayout.FieldRole, self.txtLocalidad)
self.lblDireccion = QtWidgets.QLabel(PatientDialog)
self.lblDireccion.setObjectName("lblDireccion")
self.formLayout.setWidget(13, QtWidgets.QFormLayout.LabelRole, self.lblDireccion)
self.txtDireccion = QtWidgets.QLineEdit(PatientDialog)
self.txtDireccion.setObjectName("txtDireccion")
self.formLayout.setWidget(13, QtWidgets.QFormLayout.FieldRole, self.txtDireccion)
self.lblNumeroDeTelefono1 = QtWidgets.QLabel(PatientDialog)
self.lblNumeroDeTelefono1.setObjectName("lblNumeroDeTelefono1")
self.formLayout.setWidget(14, QtWidgets.QFormLayout.LabelRole, self.lblNumeroDeTelefono1)
```

```
        self.txtNumeroDeTelefono1 = QtWidgets.QLineEdit(PatientDialog)
        self.txtNumeroDeTelefono1.setObjectName("txtNumeroDeTelefono1")
        self.formLayout.setWidget(14, QtWidgets.QFormLayout.FieldRole, self.txtNumeroDeTelefono1)
        self.lblNumeroDeTelefono2 = QtWidgets.QLabel(PatientDialog)
        self.lblNumeroDeTelefono2.setObjectName("lblNumeroDeTelefono2")
        self.formLayout.setWidget(15, QtWidgets.QFormLayout.LabelRole, self.lblNumeroDeTelefono2)
        self.txtNumeroDeTelefono2 = QtWidgets.QLineEdit(PatientDialog)
        self.txtNumeroDeTelefono2.setObjectName("txtNumeroDeTelefono2")
        self.formLayout.setWidget(15, QtWidgets.QFormLayout.FieldRole, self.txtNumeroDeTelefono2)
        self.lblCroquisVivienda_2 = QtWidgets.QLabel(PatientDialog)
        self.lblCroquisVivienda_2.setObjectName("lblCroquisVivienda_2")
        self.formLayout.setWidget(16, QtWidgets.QFormLayout.LabelRole, self.lblCroquisVivienda_2)
        self.cbCroquisVivienda = QtWidgets.QComboBox(PatientDialog)
        self.cbCroquisVivienda.setEditable(False)
        self.cbCroquisVivienda.setObjectName("cbCroquisVivienda")
        self.cbCroquisVivienda.addItem("")
        self.cbCroquisVivienda.setItemText(0, "")
        self.cbCroquisVivienda.addItem("")
        self.cbCroquisVivienda.setItemText(1, "")
        self.cbCroquisVivienda.addItem("")
        self.cbCroquisVivienda.addItem("")
        self.formLayout.setWidget(16, QtWidgets.QFormLayout.FieldRole, self.cbCroquisVivienda)
        self.spPeso = QtWidgets.QDoubleSpinBox(PatientDialog)
        self.spPeso.setDecimals(1)
        self.spPeso.setMaximum(500.0)
        self.spPeso.setObjectName("spPeso")
        self.formLayout.setWidget(7, QtWidgets.QFormLayout.FieldRole, self.spPeso)
        self.spTalla = QtWidgets.QDoubleSpinBox(PatientDialog)
        self.spTalla.setMaximum(3.0)
        self.spTalla.setObjectName("spTalla")
        self.formLayout.setWidget(8, QtWidgets.QFormLayout.FieldRole, self.spTalla)
        self.label = QtWidgets.QLabel(PatientDialog)
        self.label.setObjectName("label")
        self.formLayout.setWidget(4, QtWidgets.QFormLayout.LabelRole, self.label)
        self.textCarnet = QtWidgets.QLineEdit(PatientDialog)
        self.textCarnet.setObjectName("textCarnet")
        self.formLayout.setWidget(4, QtWidgets.QFormLayout.FieldRole, self.textCarnet)
        self.verticalLayout.addLayout(self.formLayout)
        self.buttonBox = QtWidgets.QDialogButtonBox(PatientDialog)
        self.buttonBox.setOrientation(QtCore.Qt.Horizontal)

self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Save)
        self.buttonBox.setObjectName("buttonBox")
        self.verticalLayout.addWidget(self.buttonBox)
        self.line = QtWidgets.QFrame(PatientDialog)
        self.line.setFrameShape(QtWidgets.QFrame.HLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.verticalLayout.addWidget(self.line)
        self.layLabos = QtWidgets.QVBoxLayout()
        self.layLabos.setObjectName("layLabos")
        self.verticalLayout.addLayout(self.layLabos)
        spacerItem = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem)

        self.retranslateUi(PatientDialog)
        self.buttonBox.accepted.connect(PatientDialog.accept)
        self.buttonBox.rejected.connect(PatientDialog.reject)
        QtCore.QMetaObject.connectSlotsByName(PatientDialog)

    def retranslateUi(self, PatientDialog):
        _translate = QtCore.QCoreApplication.translate
        PatientDialog.setWindowTitle(_translate("PatientDialog", "Añadir o modificar un paciente"))
        self.lblFechaConsulta.setText(_translate("PatientDialog", "Fecha consulta : "))
        self.dtFechaConsulta.setDisplayFormat(_translate("PatientDialog", "dd/MM/yyyy"))
        self.lblHClinica.setText(_translate("PatientDialog", "H.Clinica : "))
        self.lblNombre.setText(_translate("PatientDialog", "Nombre : "))
```

```
        self.lblApellido.setText(_translate("PatientDialog", "Apellido : "))
        self.lblFechaDeNacimiento.setText(_translate("PatientDialog", "Fecha de nacimiento : "))
        self.dtFechaDeNacimiento.setDisplayFormat(_translate("PatientDialog", "dd/MM/yyyy"))
        self.lblEstadoCivil_2.setText(_translate("PatientDialog", "Estado civil : "))
        self.cbEstadoCivil.setItemText(1, _translate("PatientDialog", "Casada"))
        self.cbEstadoCivil.setItemText(2, _translate("PatientDialog", "Soltera"))
        self.cbEstadoCivil.setItemText(3, _translate("PatientDialog", "Union libre"))
        self.cbEstadoCivil.setItemText(4, _translate("PatientDialog", "Divorciada"))
        self.cbEstadoCivil.setItemText(5, _translate("PatientDialog", "Viuda"))
        self.cbEstadoCivil.setItemText(6, _translate("PatientDialog", "Otros"))
        self.lblPeso.setText(_translate("PatientDialog", "Peso : "))
        self.lblTalla.setText(_translate("PatientDialog", "Talla : "))
        self.lblEstablecimientoOCentroDeSalud.setText(_translate("PatientDialog", "Establecimiento o centro de
salud : "))
        self.lblProvincia.setText(_translate("PatientDialog", "Provincia : "))
        self.lblMunicipio.setText(_translate("PatientDialog", "Municipio : "))
        self.lblLocalidad.setText(_translate("PatientDialog", "Localidad : "))
        self.lblDireccion.setText(_translate("PatientDialog", "Dirección : "))
        self.lblNumeroDeTelefono1.setText(_translate("PatientDialog", "Número de teléfono 1 : "))
        self.lblNumeroDeTelefono2.setText(_translate("PatientDialog", "Número de teléfono 2 : "))
        self.lblCroquisVivienda_2.setText(_translate("PatientDialog", "Croquis Vivienda : "))
        self.cbCroquisVivienda.setItemText(2, _translate("PatientDialog", "Si"))
        self.cbCroquisVivienda.setItemText(3, _translate("PatientDialog", "No"))
        self.spPeso.setSuffix(_translate("PatientDialog", " kg"))
        self.spTalla.setSuffix(_translate("PatientDialog", " m"))
        self.label.setText(_translate("PatientDialog", "Carnet de Identidad :"))
        self.buttonBox.setToolTip(_translate("PatientDialog", "<html><head/><body><p>Save</p></body></html>"))
        self.buttonBox.setWhatsThis(_translate("PatientDialog",
"<html><head/><body><p>Save</p></body></html>"))
```

ui_resultsdialog.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'resultsdialog.ui'
#
# Created: Sun Mar 29 09:41:01 2015
#      by: PyQt5 UI code generator 5.2.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_ResultsDialog(object):
    def setupUi(self, ResultsDialog):
        ResultsDialog.setObjectName("ResultsDialog")
        ResultsDialog.resize(657, 352)
        self.verticalLayout_2 = QtWidgets.QVBoxLayout(ResultsDialog)
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.scrollArea = QtWidgets.QScrollArea(ResultsDialog)
        self.scrollArea.setWidgetResizable(True)
        self.scrollArea.setObjectName("scrollArea")
        self.scrollAreaWidgetContents = QtWidgets.QWidget()
        self.scrollAreaWidgetContents.setGeometry(QtCore.QRect(0, 0, 637, 299))
        self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.scrollAreaWidgetContents)
        self.verticalLayout.setObjectName("verticalLayout")
        self.formLayout = QtWidgets.QFormLayout()
        self.formLayout.setObjectName("formLayout")
        self.verticalLayout.addLayout(self.formLayout)
        self.scrollArea.setWidget(self.scrollAreaWidgetContents)
        self.verticalLayout_2.addWidget(self.scrollArea)
        self.buttonBox = QtWidgets.QDialogButtonBox(ResultsDialog)
        self.buttonBox.setOrientation(QtCore.Qt.Horizontal)
```

```
self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Save)
        self.buttonBox.setObjectName("buttonBox")
        self.verticalLayout_2.addWidget(self.buttonBox)

        self.retranslateUi(ResultsDialog)
        self.buttonBox.accepted.connect(ResultsDialog.accept)
        self.buttonBox.rejected.connect(ResultsDialog.reject)
        QtCore.QMetaObject.connectSlotsByName(ResultsDialog)

    def retranslateUi(self, ResultsDialog):
        _translate = QtCore.QCoreApplication.translate
        ResultsDialog.setWindowTitle(_translate("ResultsDialog", "Añadir o modificar los datos médicos"))
```

ui_labodialog.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'labodialog.ui'
#
# Created: Sun Mar 29 09:42:05 2015
#      by: PyQt5 UI code generator 5.2.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_LaboDialog(object):
    def setupUi(self, LaboDialog):
        LaboDialog.setObjectName("LaboDialog")
        LaboDialog.resize(400, 301)
        self.buttonBox = QtWidgets.QDialogButtonBox(LaboDialog)
        self.buttonBox.setGeometry(QtCore.QRect(50, 260, 341, 32))
        self.buttonBox.setOrientation(QtCore.Qt.Horizontal)

self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Save)
        self.buttonBox.setObjectName("buttonBox")
        self.formLayoutWidget = QtWidgets.QWidget(LaboDialog)
        self.formLayoutWidget.setGeometry(QtCore.QRect(10, 60, 381, 56))
        self.formLayoutWidget.setObjectName("formLayoutWidget")
        self.formLayout = QtWidgets.QFormLayout(self.formLayoutWidget)
        self.formLayout.setFieldGrowthPolicy(QtWidgets.QFormLayout.AllNonFixedFieldsGrow)
        self.formLayout.setContentsMargins(0, 0, 0, 0)
        self.formLayout.setObjectName("formLayout")
        self.lblNombre = QtWidgets.QLabel(self.formLayoutWidget)
        self.lblNombre.setObjectName("lblNombre")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.lblNombre)
        self.txtNombre = QtWidgets.QLineEdit(self.formLayoutWidget)
        self.txtNombre.setObjectName("txtNombre")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.txtNombre)
        self.lblActivo = QtWidgets.QLabel(self.formLayoutWidget)
        self.lblActivo.setObjectName("lblActivo")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lblActivo)
        self.ChboxActivo = QtWidgets.QCheckBox(self.formLayoutWidget)
        self.ChboxActivo.setObjectName("ChboxActivo")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.ChboxActivo)

        self.retranslateUi(LaboDialog)
        self.buttonBox.accepted.connect(LaboDialog.accept)
        self.buttonBox.rejected.connect(LaboDialog.reject)
        QtCore.QMetaObject.connectSlotsByName(LaboDialog)

    def retranslateUi(self, LaboDialog):
        _translate = QtCore.QCoreApplication.translate
        LaboDialog.setWindowTitle(_translate("LaboDialog", "Añadir o modificar un laboratorio"))
        self.lblNombre.setText(_translate("LaboDialog", "Nombre : "))
```

```
        self.lblActivo.setText(_translate("LaboDialog", " Activo : "))
```

ui_testdialog.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'testdialog.ui'
#
# Created: Fri Apr  3 02:05:01 2015
#      by: PyQt5 UI code generator 5.2.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_TestDialog(object):
    def setupUi(self, TestDialog):
        TestDialog.setObjectName("TestDialog")
        TestDialog.resize(400, 300)
        self.verticalLayout = QtWidgets.QVBoxLayout(TestDialog)
        self.verticalLayout.setObjectName("verticalLayout")
        self.formLayout = QtWidgets.QFormLayout()
        self.formLayout.setFieldGrowthPolicy(QtWidgets.QFormLayout.AllNonFixedFieldsGrow)
        self.formLayout.setObjectName("formLayout")
        self.lblLaboNombre = QtWidgets.QLabel(TestDialog)
        self.lblLaboNombre.setObjectName("lblLaboNombre")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.lblLaboNombre)
        self.cbLaboNombre = QtWidgets.QComboBox(TestDialog)
        self.cbLaboNombre.setObjectName("cbLaboNombre")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.cbLaboNombre)
        self.lblNombre = QtWidgets.QLabel(TestDialog)
        self.lblNombre.setObjectName("lblNombre")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.lblNombre)
        self.txtNombre = QtWidgets.QLineEdit(TestDialog)
        self.txtNombre.setObjectName("txtNombre")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.txtNombre)
        self.lblTipo = QtWidgets.QLabel(TestDialog)
        self.lblTipo.setObjectName("lblTipo")
        self.formLayout.setWidget(2, QtWidgets.QFormLayout.LabelRole, self.lblTipo)
        self.cbTipo = QtWidgets.QComboBox(TestDialog)
        self.cbTipo.setObjectName("cbTipo")
        self.cbTipo.addItem("")
        self.cbTipo.addItem("")
        self.cbTipo.addItem("")
        self.cbTipo.addItem("")
        self.cbTipo.addItem("")
        self.formLayout.setWidget(2, QtWidgets.QFormLayout.FieldRole, self.cbTipo)
        self.lblValoresPosibles = QtWidgets.QLabel(TestDialog)
        self.lblValoresPosibles.setObjectName("lblValoresPosibles")
        self.formLayout.setWidget(3, QtWidgets.QFormLayout.LabelRole, self.lblValoresPosibles)
        self.txtValoresPosibles = QtWidgets.QLineEdit(TestDialog)
        self.txtValoresPosibles.setObjectName("txtValoresPosibles")
        self.formLayout.setWidget(3, QtWidgets.QFormLayout.FieldRole, self.txtValoresPosibles)
        self.lblActivo = QtWidgets.QLabel(TestDialog)
        self.lblActivo.setObjectName("lblActivo")
        self.formLayout.setWidget(5, QtWidgets.QFormLayout.LabelRole, self.lblActivo)
        self.ChboxActivo = QtWidgets.QCheckBox(TestDialog)
        self.ChboxActivo.setObjectName("ChboxActivo")
        self.formLayout.setWidget(5, QtWidgets.QFormLayout.FieldRole, self.ChboxActivo)
        self.chBoxMultiplesResultadoss = QtWidgets.QCheckBox(TestDialog)
        self.chBoxMultiplesResultadoss.setText("")
        self.chBoxMultiplesResultadoss.setObjectName("chBoxMultiplesResultadoss")
        self.formLayout.setWidget(4, QtWidgets.QFormLayout.FieldRole, self.chBoxMultiplesResultadoss)
        self.label = QtWidgets.QLabel(TestDialog)
        self.label.setObjectName("label")
```

```
        self.formLayout.setWidget(4, QtWidgets.QFormLayout.LabelRole, self.label)
        self.verticalLayout.addLayout(self.formLayout)
        spacerItem = QtWidgets.QSpacerItem(20, 95, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem)
        self.buttonBox = QtWidgets.QDialogButtonBox(TestDialog)
        self.buttonBox.setOrientation(QtCore.Qt.Horizontal)

self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Save)
        self.buttonBox.setObjectName("buttonBox")
        self.verticalLayout.addWidget(self.buttonBox)

        self.retranslateUi(TestDialog)
        self.buttonBox.accepted.connect(TestDialog.accept)
        self.buttonBox.rejected.connect(TestDialog.reject)
        QtCore.QMetaObject.connectSlotsByName(TestDialog)

    def retranslateUi(self, TestDialog):
        _translate = QtCore.QCoreApplication.translate
        TestDialog.setWindowTitle(_translate("TestDialog", "Añadir o modificar un test"))
        self.lblLaboNombre.setText(_translate("TestDialog", "Laboratorio: "))
        self.lblNombre.setText(_translate("TestDialog", "Nombre de la entrada:"))
        self.lblTipo.setText(_translate("TestDialog", "Tipo : "))
        self.cbTipo.setItemText(0, _translate("TestDialog", "Fecha"))
        self.cbTipo.setItemText(1, _translate("TestDialog", "Numero con decimales"))
        self.cbTipo.setItemText(2, _translate("TestDialog", "Numero entero"))
        self.cbTipo.setItemText(3, _translate("TestDialog", "Texto"))
        self.cbTipo.setItemText(4, _translate("TestDialog", "Multiopción"))
        self.lblValoresPosibles.setText(_translate("TestDialog", "Valores posibles : "))
        self.lblActivo.setText(_translate("TestDialog", "Activo :"))
        self.label.setText(_translate("TestDialog", "Multiples resultados : "))
```

<<Contrôleur>> fichiers

logindialog.py

```
from PyQt5.QtWidgets import *
from ui_logindialog import *

class LoginDialog(QDialog):
    """ The class that creates the window dialog, the first called by main.py.
        It is used to get the CI and the password of the user
    """
    def __init__(self):
        QDialog.__init__(self)

        # Create self.ui, an Ui_LoginDialog object that creates and manages the
        # different widgets contained in this dialog
        self.ui = Ui_LoginDialog()  # Create the "UI manager"
        self.ui.setupUi(self)       # And tell it to create the widgets in "self"

    def CI(self):
        """ Provides the CI written by the user in the interface
        """
        return self.ui.txtCI.text()

    def password(self):
        """ Provides the password written by the user in the interface
        """
        return self.ui.txtPassword.text()
```

mainwindow.py

```
# We will use QtWidgets, QtCore, ui_mainwindow.py, the user dialog and user.py
from PyQt5.QtWidgets import *
from ui_mainwindow import *
from ui_userdialog import *
import user
import abstractlist
import personaldialog
from exportexcel import *

# Here, we don't inherit from QDialog but from QMainWindow, because we have told
# QtDesigner that we wanted to have a main window (in order to have a menu bar)
class MainWindow(QMainWindow):
    """ Class that displays the main window
    """
    def __init__(self, ID):
        QMainWindow.__init__(self)
        currentusu = user.getUser(ID)

        # Create self.ui, an Ui_MainWindow object that creates and manages the
        # different widgets contained in this window
        self.ui = Ui_MainWindow()   # Create the "UI manager"
        self.ui.setupUi(self)       # And tell it to create the widgets in "self"

        # Connect the slots that we need
        self.ui.mnuQuit.triggered.connect(self.close)
        self.ui.pushButton_2.clicked.connect(self.changePersonal)
        self.ui.btnDesconectarses.clicked.connect(self.close)
        self.ui.btnExcel.clicked.connect(self.exportexcel)
        self.ui.btnVeronique.clicked.connect(self.statistics)

    def notAdmin(self):
        """ Removes the tab Usuarios if the user has not the privilege of
            managing the users
        """
        self.ui.tabWidget.removeTab(0)   # File » Quit closes the window

    def changePersonal(self):
        """ Displays the PersonalDialog when the button <Completar los datos
            personales> is pushed.
        """
        dlg = personaldialog.PersonalDialog()
        dlg.display()
        if dlg.exec_() == QDialog.Accepted:
                dlg.update()

    def notModif(self,i):
        """ Removes the tabs Laboratorios and Entradas if the user has not the
            privilege of modifing the database. The i variable is added to
            avoid a shift in the removing tabs, regarding whether the Usuarios
            tab has been removed or not
        """

        self.ui.tabWidget.removeTab(2+i)
        self.ui.tabWidget.removeTab(1+i)

    def exportexcel(self):
        """ Creates the csv file that will contain the columes that sumarizes
            the personal and medical data of the patients who have been
            inserted or modified during the period specified in the interface
        """
```

```
        exportexcel(self.ui.dtDel.date(),self.ui.dtAl.date())

    def statistics(self):
        """ Creates a csv file that will contain the medical information
            required by professor Veronique Fontaine to do his statistics
        """
        exportstats()
```

userdialog.py

```
# We will use QtWidgets, the Ui_UserDialog class and the User class in user.py
from PyQt5.QtWidgets import *

from ui_userdialog import *
from user import *
import random
import hashlib
import id

class UserDialog(QDialog):
    """ This class contains the methods that can be applied on a dialog box
        that opens when a user want to add or modify another user, specifying
        some of his personal information, his password and his privileges
    """

    def __init__(self, parent):
        """ The constructor initializes an Ui_UserDialog object to create the
            interface where the information of the user will be displayed and
            modified.
        """
        QDialog.__init__(self, parent)

        # Create self.ui, an Ui_UserDialog object that creates and manages the
        # different widgets contained in this dialog
        self.ui = Ui_UserDialog()   # Create the "UI manager"
        self.ui.setupUi(self)       # And tell it to create the widgets in "self"
        self.ui.crearpass.clicked.connect(self.newPassword)
        self.ui.buttons.accepted.connect(self.validate)


    # When we edit an existing user, pre-populate the fields with the values
    # already in the database. The user parameter of this method is an User instance

    def displayObject(self, user):
        """ Displays the information of the user if this is wanted to be
        modified. Wheter it is a new one, it fills with none information
        """
        self.ui.txtFirstName.setText(user.nombre())
        self.ui.txtLastName.setText(user.apellido())
        self.ui.txtCI.setText(user.CI())
        self.ui.txtCI.setEnabled(False)
        self.disp_privilegios(user)
      # Password empty

    # The user has possibly modified information about an user, now we would
    # like to copy the contents of the fields back into an User object
    def updateObject(self, user):
        """ Updates the information of the user being modified
        """
        user.setNombre(self.ui.txtFirstName.text())
        user.setApellido(self.ui.txtLastName.text())
        user.setCI(self.ui.txtCI.text())
```

```
        user.setPassword(self.ui.txtPassword.text())

        self.privilegios(user)

    def disp_privilegios(self,user):
        """ Acording to the checkboxes marked, displays the privileges of the
        user and his category, associating each group to a set of privilege
        +category. Each privilege corresponds to one checkbox and the category
        markes just one of the four radiobuttons
        """
        if user.grupo() == 1:
            self.ui.radioButton.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(2)
            self.ui.checkBox_gestio.setCheckState(2)
        if user.grupo() == 2:
            self.ui.radioButton_2.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(2)
            self.ui.checkBox_gestio.setCheckState(2)
        if user.grupo() == 3:
            self.ui.radioButton_3.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(2)
            self.ui.checkBox_gestio.setCheckState(2)
        if user.grupo() == 4:
            self.ui.radioButton_4.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(2)
            self.ui.checkBox_gestio.setCheckState(2)
        if user.grupo() == 5:
            self.ui.radioButton.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(0)
            self.ui.checkBox_gestio.setCheckState(2)
        if user.grupo() == 6:
            self.ui.radioButton_2.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(0)
            self.ui.checkBox_gestio.setCheckState(2)
        if user.grupo() == 7:
            self.ui.radioButton_3.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(0)
            self.ui.checkBox_gestio.setCheckState(2)
        if user.grupo() == 8:
            self.ui.radioButton_4.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(0)
            self.ui.checkBox_gestio.setCheckState(2)
        if user.grupo() == 9:
            self.ui.radioButton.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(2)
            self.ui.checkBox_gestio.setCheckState(0)
        if user.grupo() == 10:
            self.ui.radioButton_2.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(2)
            self.ui.checkBox_gestio.setCheckState(0)
        if user.grupo() == 11:
            self.ui.radioButton_3.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(2)
            self.ui.checkBox_gestio.setCheckState(0)
        if user.grupo() == 12:
            self.ui.radioButton_4.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(2)
            self.ui.checkBox_gestio.setCheckState(0)
        if user.grupo() == 13:
            self.ui.radioButton.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(0)
            self.ui.checkBox_gestio.setCheckState(0)
        if user.grupo() == 14:
            self.ui.radioButton_2.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(0)
            self.ui.checkBox_gestio.setCheckState(0)
        if user.grupo() == 15:
            self.ui.radioButton_3.setChecked(True)
```

```
            self.ui.checkBox_Modif.setCheckState(0)
            self.ui.checkBox_gestio.setCheckState(0)
        if user.grupo() == 16:
            self.ui.radioButton_4.setChecked(True)
            self.ui.checkBox_Modif.setCheckState(0)
            self.ui.checkBox_gestio.setCheckState(0)

    def privilegios(self,user):
        """ From the category marked and the checkboxes of the privileges
        filled, it associates the user to a certain group
        """
        if self.ui.checkBox_Modif.checkState() == 2 and self.ui.checkBox_gestio.checkState() == 2:
            if self.ui.radioButton.isChecked():
                user.setGrupo(1)
            elif self.ui.radioButton_2.isChecked():
                user.setGrupo(2)
            elif self.ui.radioButton_3.isChecked():
                user.setGrupo(3)
            elif self.ui.radioButton_4.isChecked():
                user.setGrupo(4)
        elif self.ui.checkBox_Modif.checkState() == 0 and self.ui.checkBox_gestio.checkState() == 2:
            if self.ui.radioButton.isChecked():
                user.setGrupo(5)
            elif self.ui.radioButton_2.isChecked():
                user.setGrupo(6)
            elif self.ui.radioButton_3.isChecked():
                user.setGrupo(7)
            elif self.ui.radioButton_4.isChecked():
                user.setGrupo(8)
        elif self.ui.checkBox_Modif.checkState() == 2 and self.ui.checkBox_gestio.checkState() == 0:
            if self.ui.radioButton.isChecked():
                user.setGrupo(9)
            elif self.ui.radioButton_2.isChecked():
                user.setGrupo(10)
            elif self.ui.radioButton_3.isChecked():
                user.setGrupo(11)
            elif self.ui.radioButton_4.isChecked():
                user.setGrupo(12)
        if self.ui.checkBox_Modif.checkState() == 0 and self.ui.checkBox_gestio.checkState() == 0:
            if self.ui.radioButton.isChecked():
                user.setGrupo(13)
            elif self.ui.radioButton_2.isChecked():
                user.setGrupo(14)
            elif self.ui.radioButton_3.isChecked():
                user.setGrupo(15)
            elif self.ui.radioButton_4.isChecked():
                user.setGrupo(16)


    def newPassword(self):
        """ Gives to the user being modfied or added a new password
        """
        pswd = random.randrange(1,1000000,6)
        hshpswd = hashlib.sha1(bytes(pswd, encoding='utf8')).hexdigest()
        self.ui.txtPassword.setText(str(pswd))



    def validate(self):
        """ Validates whether a CI, a name, a lastname and a category has been
        introduced. If not, it displays a text at the bottom of the interface
        that indicates that some field is missing
        """
        if self.ui.txtFirstName.text() != '' and self.ui.txtLastName.text() != '' and self.ui.txtCI.text() != '':
            if self.ui.radioButton.isChecked() or self.ui.radioButton_2.isChecked() or self.ui.radioButton_3.isChecked()
or self.ui.radioButton_4.isChecked():
                self.accept()
            else:
```

```
            self.ui.labelError2.setText('Indique la categoria del nuevo utilizador')

        else:
            self.ui.labelError1.setText('No ha rellenado todos los campos')
```

personaldialog.py

```
from PyQt5.QtWidgets import *
from ui_personaldialog import *
import id
import user

class PersonalDialog(QDialog):
    """ This class contains the methods that can be applied on a dialog box that
        opens when the user wants to add or modify his personal data
    """

    def __init__(self):
        """ The constructor initializes an Ui_PatientDialog object to create
            the user's interface
        """
        QDialog.__init__(self)
        self.ui = Ui_PersonalDialog()
        self.ui.setupUi(self)
        self.ui.buttons.accepted.connect(self.validate)

    def current_u(self):
        """ Provides the user that is currently using the database
        """
        return user.getUser(id.current_id())

    def display(self):
        """ Displays the information of the user in the interface and
            disenables the CI to be modified
        """
        u = self.current_u()
        self.ui.txtNombre.setText(u.nombre())
        self.ui.txtApellido.setText(u.apellido())
        self.ui.txtCI.setText(u.CI())
        self.ui.txtCI.setEnabled(False)
        self.ui.dateEdit.setDate(u.nacimiento())
        self.ui.txtTelef.setText(u.telefono())
        self.ui.txtEmail.setText(u.email())
        self.ui.txtDomicilio.setText(u.domicilio())

    def update(self):
        """ Updates the information that the user has changed in the interace
        """
        u = self.current_u()
        u.setNombre(self.ui.txtNombre.text())
        u.setApellido(self.ui.txtApellido.text())
        u.setNacimiento(self.ui.dateEdit.date())
        u.setTelefono(self.ui.txtTelef.text())
        u.setEmail(self.ui.txtEmail.text())
        u.setDomicilio(self.ui.txtDomicilio.text())
        u.setPassword(self.ui.txtPassword.text())
        u.save(0)

    def validate(self):
        if self.ui.txtPassword.text()!=self.ui.txtPassword2.text():
            self.ui.lblError.setText('Contraseña mal confirmada')
        else:
            self.accept()
```

patientdialog.py

```
from PyQt5.QtWidgets import *
from ui_patientdialog import *
from resultsdialog import *
from patient import *
import id

class PatientDialog(QDialog):
    """ This class contains the methods that can be applied on a dialog box that
        opens when the user wants to add or modify a patient and its personal
        data. This dialog box is a PatientDialog object. It also contains buttons
        corresponding to the different laboratories.
    """
    def __init__(self, parent):
        """ The constructor initializes an Ui_PatientDialog object to create a
            user interface. It takes into account that everybody cannot modify
            the personal data of a patient. It also initializes the buttons
            corresponding to the different laboratories and the associated
            results.
        """
        QDialog.__init__(self, parent)

        usr = id.current_user()

        # Create the user interface
        self.ui = Ui_PatientDialog()
        self.ui.setupUi(self)
        if 0 not in usr.allowedPatientList():
            self.ui.dtFechaConsulta.setEnabled(False)
            self.ui.txtHClinica.setEnabled(False)
            self.ui.txtNombre.setEnabled(False)
            self.ui.txtApellido.setEnabled(False)
            self.ui.textCarnet.setEnabled(False)
            self.ui.dtFechaDeNacimiento.setEnabled(False)
            self.ui.cbEstadoCivil.setEnabled(False)
            self.ui.spPeso.setEnabled(False)
            self.ui.spTalla.setEnabled(False)
            self.ui.txtEstablecimientoOCentroDeSalud.setEnabled(False)
            self.ui.txtProvincia.setEnabled(False)
            self.ui.txtMunicipio.setEnabled(False)
            self.ui.txtLocalidad.setEnabled(False)
            self.ui.txtDireccion.setEnabled(False)
            self.ui.txtNumeroDeTelefono1.setEnabled(False)
            self.ui.txtNumeroDeTelefono2.setEnabled(False)
            self.ui.cbCroquisVivienda.setEnabled(False)

        # List the laboratories from the database and create buttons for them
        query = db.Query()
        query.prepare("SELECT id,Nombre FROM laboratorios WHERE Activo = 1")
        query.exec_()


        while query.next():
            # Create a new button
            button = QPushButton(query.value(1).replace('_',' ') + "...", self)

            # Connect its clicked slot
            button.clicked.connect(self.makeSlotForLabo(query.value(0)))

            # Add the button to the layout
            self.ui.layLabos.addWidget(button)

    def makeSlotForLabo(self, labo_id):
        """ Called when clicking on a button corresponding to a certain
            laboratory. It calls itself the method laboClicked.
        """
```

```
        # lambda is used to stock labo_id inside a fonction that does
        # any parameter, due to the incapability of a button of
        # providing a parameter to his slote
        return lambda: self.laboClicked(labo_id)

    def laboClicked(self, labo_id):
        """ Called by the method makeSlotForLabo to open a ResultsDialog object.
            The results entered by the user will be saved in the database.
        """

        # self.patient_id contains the ID of the patient being displayed and
        # labo_id contains the ID of the labo on which the user click
        dlg = ResultsDialog(self,labo_id,self.patient_id)

        # Open the dialog, let the user interact with it, and if he/she clicks on
        # Ok, then save the updated results in the database
        if dlg.exec_() == QDialog.Accepted:
            dlg.saveInDatabase()

    def displayObject(self, patient):
        """ Sets the attributes of the Patient object <patient> in the dialog
            box.
        """
        self.ui.dtFechaConsulta.setDate(patient.fecha_consulta())
        self.ui.txtHClinica.setText(patient.h_Clinica())
        self.ui.txtNombre.setText(patient.nombre())
        self.ui.txtApellido.setText(patient.apellido())
        self.ui.textCarnet.setText(patient.carnet_de_identidad())
        self.ui.dtFechaDeNacimiento.setDate(patient.fecha_de_nacimiento())
        self.ui.cbEstadoCivil.setCurrentIndex(patient.estado_civil())
        self.ui.spPeso.setValue(patient.peso())
        self.ui.spTalla.setValue(patient.talla())
        self.ui.txtEstablecimientoOCentroDeSalud.setText(patient.establecimiento_o_centro_de_salud())
        self.ui.txtProvincia.setText(patient.provincia())
        self.ui.txtMunicipio.setText(patient.municipio())
        self.ui.txtLocalidad.setText(patient.localidad())
        self.ui.txtDireccion.setText(patient.direccion())
        self.ui.txtNumeroDeTelefono1.setText(patient.numero_de_telefono_1())
        self.ui.txtNumeroDeTelefono2.setText(patient.numero_de_telefono_2())
        self.ui.cbCroquisVivienda.setCurrentIndex(patient.croquis_vivienda())


        self.patient_id = patient.id()

    def updateObject(self, patient):
        """ Associates the data entered by the user in the dialog box to the
            attributes of the Patient object <patient>.
        """
        patient.setFecha_consulta(self.ui.dtFechaConsulta.date())
        patient.setH_Clinica(self.ui.txtHClinica.text())
        patient.setNombre(self.ui.txtNombre.text())
        patient.setApellido(self.ui.txtApellido.text())
        patient.setCarnet_de_identidad(self.ui.textCarnet.text())
        patient.setFecha_de_nacimiento(self.ui.dtFechaDeNacimiento.date())
        patient.setEstado_civil(self.ui.cbEstadoCivil.currentIndex())
        patient.setPeso(self.ui.spPeso.value())
        patient.setTalla(self.ui.spTalla.value())
        patient.setEstablecimiento_o_centro_de_salud(self.ui.txtEstablecimientoOCentroDeSalud.text())
        patient.setProvincia(self.ui.txtProvincia.text())
        patient.setMunicipio(self.ui.txtMunicipio.text())
        patient.setLocalidad(self.ui.txtLocalidad.text())
        patient.setDireccion(self.ui.txtDireccion.text())
        patient.setNumero_de_telefono_1(self.ui.txtNumeroDeTelefono1.text())
        patient.setNumero_de_telefono_2(self.ui.txtNumeroDeTelefono2.text())
        patient.setCroquis_vivienda(self.ui.cbCroquisVivienda.currentIndex())
```

resultsdialog.py

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import db

from ui_resultsdialog import *
from resultwidget import *

import result
import id


class ResultsDialog(QDialog):
    """ This class contains the methods that can be applied on a dialog box that
        opens when the user wants to add or modify a result of a patient in a
        certain laboratory. This dialog box is a ResultsDialog object.
    """
    def __init__(self, parent, Labo_id, Paciente_id):
        """ The constructor initializes a Ui_ResultsDialog object to create a
            user interface. It also initializes the id's of the patient and the
            laboratory and a widget is created in function of those attributes.
            This widget contains all the results for the given patient and
            laboratory. It takes into account that everybody cannot modify all
            the results.
        """
        QDialog.__init__(self, parent)

        # Create the user interface
        self.ui = Ui_ResultsDialog()
        self.ui.setupUi(self)

        self._Labo_id = Labo_id
        self._Paciente_id = Paciente_id

        self.usr = id.current_user()

        # Dictionary that associates each exam with a ResultWidget
        self._result_widgets = {}

        for r in result.allResults(self.labo_id(), self.paciente_id()):
            if r.examen_id() not in self._result_widgets:
                # Create a new ResultWidget for this exam that we see for the
                # first time
                widget = ResultWidget(self)
                widget.setEnabled(self.labo_id() in self.usr.allowedPatientList())

                self._result_widgets[r.examen_id()] = widget
                self.ui.formLayout.addRow(QLabel(r.nombre().replace('_',' '), self), widget)

            # Add the result to the result widget
            self._result_widgets[r.examen_id()].addResult(r)

    """ The following methods are used to get the attributes of a ResultsDialog
        object.
    """
    def labo_id(self):
        return self._Labo_id

    def paciente_id(self):
        return self._Paciente_id

    def saveInDatabase(self):
        """ Iterates through all the results and updates them in the database
            using what the user has entered in their widget.
        """
        for widget in self._result_widgets.values():
```

```
        r = widget.result()

        # A new result already knows its lab and exam but not yet its patient,
        # so set this information here. If the result is already in the
        # database, this line causes no harm because it will replace the
        # patient id with itself
        if r is not None:
            r.setPaciente_id(self.paciente_id())

            r.save(1)
```

labodialog.py

```
from PyQt5.QtWidgets import *

from ui_labodialog import *
from labo import *

class LaboDialog(QDialog):
    """ This class contains the methods that can be applied on a dialog box that
        opens when the user wants to add or modify a laboratory. This dialog box
        is a LaboDialog object.
    """
    def __init__(self, parent):
        """ The constructor initializes an Ui_LaboDialog object to create a
            user interface.
        """
        QDialog.__init__(self, parent)

        self.ui = Ui_LaboDialog()
        self.ui.setupUi(self)

    def displayObject(self, labo):
        """ Sets the attributes of the Labo object <labo> in the dialog box.
        """
        self.ui.txtNombre.setText(labo.nombre().replace('_',' '))

        # This QCheckBox, whether is checked or not, defines if the button of the
        # laboratory will appear or not at PatientDialog
        self.ui.ChboxActivo.setChecked(labo.activo())


    def updateObject(self, labo):
        """ Associates the data entered by the user in the dialog box to the
            attributes of the Labo object <labo>.
        """
        # The replace is necessary as the columns of the table 'examenes' of the
        # database cannot have gaps in his definition
        labo.setNombre(self.ui.txtNombre.text().replace(' ','_'))
        labo.setActivo(self.ui.ChboxActivo.isChecked())
```

testdialog.py

```
from PyQt5.QtWidgets import *

from ui_testdialog import *
from test import *
import labo
import id
```

```
class TestDialog(QDialog):
    """ This class contains the methods that can be applied on a dialog box that
        opens when the user wants to add or modify a test. This dialog box is a
        TestDialog object.
    """
    def __init__(self, parent):
        """ The constructor initializes a Ui_TestDialog object to create a
            user interface. It also takes into account that everybody cannot
            modify all the tests.
        """
        QDialog.__init__(self, parent)

        self.ui = Ui_TestDialog()
        self.ui.setupUi(self)
        usr = id.current_user()
        for l in labo.allLabos(''):
            if l.id() in usr.allowedLabosModifs():
                self.ui.cbLaboNombre.addItem(l.nombre().replace('_',' '), l.id())

    def displayObject(self, test):
        """ Sets the attributes of the Test object <test> in the dialog box.
        """
        # It explores the elements of the ComboBox and selects the one with
        # the correct ID.

        # #count(): This property holds the number of items in the combobox,
        # by default, for an empty combo box, this property has a value of 0.
        for i in range(self.ui.cbLaboNombre.count()):

            # itemData(): Returns the data for the given role in the given
            # index in the combobox, or QVariant::Invalid if there is no
            # data for this role.
            if self.ui.cbLaboNombre.itemData(i) == test.labo_id():
                self.ui.cbLaboNombre.setCurrentIndex(i)

        self.ui.txtNombre.setText(test.nombre().replace('_',' '))
        self.ui.cbTipo.setCurrentIndex(test.tipo())
        self.ui.txtValoresPosibles.setText(test.valores_posibles())
        self.ui.chBoxMultiplesResultadoss.setChecked(test.multiples_resultados())
        self.ui.ChboxActivo.setChecked(test.activo())


    def updateObject(self, test):
        """ Associates the data entered by the user in the dialog box to the
            attributes of the Test object <test>.
        """
        test.setLabo_id(self.ui.cbLaboNombre.currentData())
        test.setNombre(self.ui.txtNombre.text().replace(' ','_'))
        test.setTipo(self.ui.cbTipo.currentIndex())
        test.setValores_posibles(self.ui.txtValoresPosibles.text())
        test.setActivo(self.ui.ChboxActivo.isChecked())
        test.setMultiples_resultados(self.ui.chBoxMultiplesResultadoss.isChecked())
```

<<Modèle>> fichiers

user.py

```
import db
import hashlib
import random

class User(object):
    """ Class which contains all the methods and attributes that can be applied
        to a user
```

```
    """
    def __init__(self, ID, apellido, nombre, nacimiento, CI, grupo, telefono, email, domicilio):
        self._apellido = apellido
        self._nombre = nombre
        self._CI = CI
        self._nacimiento = nacimiento
        self._ID = ID
        self._password = None
          self._grupo = grupo
        self._telefono = telefono
        self._email = email
        self._domicilio = domicilio


    """ The following functions are used to get the information of the user
    """
    def apellido(self):
        return self._apellido

    def nombre(self):
        return self._nombre

    def CI(self):
        return self._CI

    def nacimiento(self):
        return self._nacimiento

    def ID(self):
        return self._ID

    def grupo(self):
        return self._grupo

    def telefono(self):
        return self._telefono

    def email(self):
        return self._email

    def domicilio(self):
        return self._domicilio

    def password(self):
        return self._password

    def allowedLabosModifs(self):
        """ Returns a dictionary that associates for each group, the IDs of the
        laboratories that can be modified by a member of the group
        """
        allowed_labosModifs = {
            1:  [1],
            2:  [2, 4],
            3:  [3],
            4:  [2],
            5:  [],
            6:  [],
            7:  [],
            8:  [],
            9:  [1],
            10: [2, 4],
            11: [3],
            12: [2],
            13: [],
            14: [],
            15: [],
            16: []
        }
```

```
    return allowed_labosModifs[self.grupo()]

def allowedPatientList(self):
    """ Returns a dictionary that associates for each group, the IDs of the
    laboratory's tests that can be modified by a member of the group,
    being 0 the personal data
    """

    allowed_patientList = {
        1: [0, 1],
        2: [0, 2, 4],
        3: [3],
        4: [2],
        5: [0, 1],
          6: [0, 2, 4],
        7: [3],
          8: [2],
          9: [0, 1],
          10: [0, 2, 4],
          11: [3],
          12: [2],
          13: [0, 1],
        14: [0, 2, 4],
        15: [3],
        16: [2]
    }
    return allowed_patientList[self.grupo()]


""" The following functions are used to set the information of the user
"""
def setNombre(self, value):
    self._nombre = value

def setApellido(self, value):
    self._apellido = value

def setCI(self,value):
    self._CI = value

def setID(self, value):
    self._ID = value

def setNacimiento(self, value):
    self._nacimiento = value

def setPassword(self, value):
    self._password = value

def setGrupo(self, value):
    self._grupo = value

def setTelefono(self, value):
    self._telefono = value

def setEmail(self, value):
    self._email = value

def setDomicilio(self, value):
    self._domicilio = value

def save(self, option):
    """ Saves the information  of the user when his information is modified
    in the interfaces personaldialog and userdialog
    """
    query = db.Query()
    query.prepare("UPDATE usuarios SET Apellido=?, Nombre=?, Nacimiento=?,CI=?, Grupo=?, Telefono=?,
```

```
Email=?, Domicilio=? WHERE ID=?")
        query.addBindValue(self.apellido())
        query.addBindValue(self.nombre())
        query.addBindValue(self.nacimiento())
        query.addBindValue(self.CI())
        query.addBindValue(self.grupo())
        query.addBindValue(self.telefono())
        query.addBindValue(self.email())
        query.addBindValue(self.domicilio())
        query.addBindValue(int(self.ID()))
        query.exec_()

        if option == 0:
            if self.password()!="":
                query.exec_("SET PASSWORD = PASSWORD('%s');" % self.password().replace("'", "''"))
                query.exec_()

        else:
            query.prepare("SELECT FROM usuarios WHERE ID=?;")
            query.addBindValue(self.ID())

            if query.exec_():
                query.exec_("SET PASSWORD FOR %s = PASSWORD('%s')" % (self.CI().replace("'", "''"),
self.password().replace("'", "''")))
            else:

                query.exec_("CREATE USER '%s' IDENTIFIED BY '%s';" % (self.CI().replace("'", "''"),
self.password().replace("'", "''")))

                query.exec_("GRANT SELECT ON *.* TO '%s';" % self.CI().replace("'", "''"))
                query.exec_("GRANT INSERT, DELETE, UPDATE ON codepo.pacientes TO '%s';" %
self.CI().replace("'", "''"))
                query.exec_("GRANT INSERT, DELETE, UPDATE ON codepo.resultados TO '%s';" %
self.CI().replace("'", "''"))

                ss = [1,2,3,4,5,6,7,8]
                jj = [1,2,3,4,9,10,11,12]

                if self.grupo() in ss:
                    query.exec_("GRANT CREATE USER ON *.* TO '%s' WITH GRANT OPTION;" %
self.CI().replace("'", "''"))
                    query.exec_("GRANT INSERT, DELETE, UPDATE ON codepo.usuarios TO '%s';" %
self.CI().replace("'", "''"))

                if self.grupo() in jj:
                    query.exec_("GRANT SELECT, DELETE, UPDATE, INSERT ON codepo.laboratorios TO '%s';" %
self.CI().replace("'","''"))
                    query.exec_("GRANT SELECT, DELETE, UPDATE, INSERT ON codepo.examenes TO '%s';" %
self.CI().replace("'","''"))


def login(CI, password):
    """ Connects for the first time to the database and checks if the entered
    user and password correspond to one of the users of the database
    """

    # Connect to the database
    try:
        db.connect(CI, password)
    except:
        return (False,None)

    # Identify the user
    query = db.Query()
    query.prepare("SELECT ID FROM usuarios WHERE CI=?;")
    query.addBindValue(CI)
    query.exec_()
    return (query.next(),query.value(0))
```

```
def allUsers(entry):
    """ Returns a list with all the users from the database including <entry> in
        their name, first name or id. This parameter can be empty.
    """
    allU = []
    query = db.Query()
    query.prepare("SELECT ID, Apellido, Nombre, Nacimiento, CI, Grupo, Telefono, Email, Domicilio FROM
usuarios WHERE Nombre LIKE ? OR Apellido LIKE ? OR CI LIKE ?")
    query.addBindValue('%' + entry + '%')
    query.addBindValue('%' + entry + '%')
    query.addBindValue('%' + entry + '%')
    query.exec_()
    while query.next():
        allU.append(User(int(query.value(0)), query.value(1), query.value(2), query.value(3), query.value(4),
query.value(5), query.value(6), query.value(7), query.value(8)))
    return allU

def getUser(ID):
    """ Provides the database user corresponding to the given <id>.
    """
    query = db.Query()
    query.prepare("SELECT Apellido, Nombre, Nacimiento, CI, Grupo, Telefono, Email, Domicilio FROM usuarios
WHERE ID=?")
    query.addBindValue(ID)
    query.exec_()
    if query.next():
        return User(ID,query.value(0), query.value(1), query.value(2), query.value(3), query.value(4), query.value(5),
query.value(6), query.value(7))
    else:
        pass

def createUser():
    """ Creates a new user in the database with any information exept the ID,
        which will be the last user ID +1. And it returns it
    """
    query = db.Query()
    query.prepare("INSERT INTO usuarios(Apellido, Nombre, Nacimiento, CI, Grupo, Telefono, Email, Domicilio)
VALUES('','','',0,'','','')")
    query.exec_()
    query.lastInsertId()
    usr = User(int(query.lastInsertId()),'','','',0,'','','')
    return usr

def removeUser(ID):
    """ Removes the user from the database with the corresponding ID
    """
    query = db.Query()
    u = getUser(ID)

    query.prepare("DELETE FROM usuarios WHERE ID=?")
    query.addBindValue(ID)
    query.exec_()
    query.exec_("DROP USER '%s';" % u.CI().replace("'", ""))
```

patient.py

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import db

class Patient(object):
    """ This class contains the attributes and methods that can be applied on a
        patient.
    """
    def __init__(self, id, Fecha_consulta, H_Clinica, Nombre, Apellido,
```

```
            Carnet_de_identidad, Fecha_de_nacimiento, Estado_civil, Peso,
            Talla, Establecimiento_o_centro_de_salud, Provincia, Municipio,
            Localidad, Direccion, Numero_de_telefono_1,
            Numero_de_telefono_2, Croquis_vivienda):
        """ The constructor initializes the attributes of a patient with the
            values given as parameters.
        """

        self._id = id
        self._Fecha_consulta = Fecha_consulta
        self._H_Clinica = H_Clinica
        self._Carnet_de_identidad = Carnet_de_identidad
        self._Nombre = Nombre
        self._Apellido = Apellido
        self._Fecha_de_nacimiento = Fecha_de_nacimiento
        self._Estado_civil = Estado_civil
        self._Peso = Peso
        self._Talla = Talla
        self._Establecimiento_o_centro_de_salud = Establecimiento_o_centro_de_salud
        self._Provincia = Provincia
        self._Municipio = Municipio
        self._Localidad = Localidad
        self._Direccion = Direccion
        self._Numero_de_telefono_1 = Numero_de_telefono_1
        self._Numero_de_telefono_2 = Numero_de_telefono_2
        self._Croquis_vivienda = Croquis_vivienda

    """ The following methods are used to get the attributes of a patient.
    """
    def id(self):
        return self._id

    def fecha_consulta(self):
        return self._Fecha_consulta

    def h_Clinica(self):
        return self._H_Clinica

    def nombre(self):
        return self._Nombre

    def apellido(self):
        return self._Apellido

    def carnet_de_identidad(self):
        return self._Carnet_de_identidad

    def fecha_de_nacimiento(self):
        return self._Fecha_de_nacimiento

    def estado_civil(self):
        return self._Estado_civil

    def peso(self):
        return self._Peso

    def talla(self):
        return self._Talla

    def establecimiento_o_centro_de_salud(self):
        return self._Establecimiento_o_centro_de_salud

    def provincia(self):
        return self._Provincia

    def municipio(self):
        return self._Municipio
```

```
    def localidad(self):
        return self._Localidad

    def direccion(self):
        return self._Direccion

    def numero_de_telefono_1(self):
        return self._Numero_de_telefono_1

    def numero_de_telefono_2(self):
        return self._Numero_de_telefono_2

    def croquis_vivienda(self):
        return self._Croquis_vivienda


    """ The following methods are used to set <value> as the value of an
        attribute of a patient.
    """
    def setFecha_consulta(self, value):
        self._Fecha_consulta = value

    def setH_Clinica(self, value):
        self._H_Clinica = value

    def setNombre(self, value):
        self._Nombre = value

    def setApellido(self, value):
        self._Apellido = value

    def setCarnet_de_identidad(self, value):
        self.Carnet_de_identidad = value

    def setFecha_de_nacimiento(self, value):
        self._Fecha_de_nacimiento = value

    def setEstado_civil(self, value):
        self._Estado_civil = value

    def setPeso(self, value):
        self._Peso = value

    def setTalla(self, value):
        self._Talla = value

    def setEstablecimiento_o_centro_de_salud(self, value):
        self._Establecimiento_o_centro_de_salud = value

    def setProvincia(self, value):
        self._Provincia = value

    def setMunicipio(self, value):
        self._Municipio = value

    def setLocalidad(self, value):
        self._Localidad = value

    def setDireccion(self, value):
        self._Direccion = value

    def setNumero_de_telefono_1(self, value):
        self._Numero_de_telefono_1 = value

    def setNumero_de_telefono_2(self, value):
        self._Numero_de_telefono_2 = value

    def setCroquis_vivienda(self, value):
```

```
        self._Croquis_vivienda = value

    def save(self,option):
        """ Updates, in the database, the attributes of a database patient added
            or modified by the user in patientdialog. Remind that, when adding a
            patient, it is saved in the database with empty attributes by
            default.
        """
        query = db.Query()
        query.prepare("UPDATE pacientes SET Fecha_consulta = ?,H_Clinica =?,Nombre =?,Apellido
=?,Carnet_de_Identidad =?, Fecha_de_nacimiento =?,Estado_civil =?,Peso =?,Talla
=?,Establecimiento_o_centro_de_salud =?,Provincia =?,Municipio =?,Localidad =?,Direccion
=?,Numero_de_telefono_1 =?,Numero_de_telefono_2 =?,Croquis_vivienda =?, Fecha_de_insercion =
CURRENT_DATE() WHERE id=?")
        query.addBindValue(self.fecha_consulta())
        query.addBindValue(self.h_Clinica())
        query.addBindValue(self.nombre())
        query.addBindValue(self.apellido())
        query.addBindValue(self.carnet_de_identidad())
        query.addBindValue(self.fecha_de_nacimiento())
        query.addBindValue(self.estado_civil())
        query.addBindValue(self.peso())
        query.addBindValue(self.talla())
        query.addBindValue(self.establecimiento_o_centro_de_salud())
        query.addBindValue(self.provincia())
        query.addBindValue(self.municipio())
        query.addBindValue(self.localidad())
        query.addBindValue(self.direccion())
        query.addBindValue(self.numero_de_telefono_1())
        query.addBindValue(self.numero_de_telefono_2())
        query.addBindValue(self.croquis_vivienda())
        query.addBindValue(self.id())
        query.exec_()

def allPatients(entry):
    """ Returns a list with all the patients from the database including <entry>
        in their name, first name or id. This parameter can be empty.
    """
    query = db.Query()
    query.prepare("SELECT
id,Fecha_consulta,H_Clinica,Nombre,Apellido,Carnet_de_Identidad,Fecha_de_nacimiento,Estado_civil,Peso,Tall
a,Establecimiento_o_centro_de_salud,Provincia,Municipio,Localidad,Direccion,Numero_de_telefono_1,Numero_
de_telefono_2,Croquis_vivienda  FROM pacientes WHERE Nombre LIKE ? OR Apellido LIKE ? OR ID LIKE ?")
    query.addBindValue('%' + entry + '%')
    query.addBindValue('%' + entry + '%')
    query.addBindValue('%' + entry + '%')
    query.exec_()
    patients = []
    while query.next():

patients.append(Patient(query.value(0),query.value(1),query.value(2),query.value(3),query.value(4),query.value(5
),query.value(6),query.value(7),query.value(8),query.value(9),query.value(10),query.value(11),query.value(12),qu
ery.value(13),query.value(14),query.value(15),query.value(16), query.value(17)))
    return patients

def getPatient(id):
    """ Returns the database patient corresponding to the given <id>.
    """
    query = db.Query()
    query.prepare("SELECT
id,Fecha_consulta,H_Clinica,Nombre,Apellido,Carnet_de_Identidad,Fecha_de_nacimiento,Estado_civil,Peso,Tall
a,Establecimiento_o_centro_de_salud,Provincia,Municipio,Localidad,Direccion,Numero_de_telefono_1,Numero_
de_telefono_2,Croquis_vivienda FROM pacientes WHERE id=?")
    query.addBindValue(id)
    query.exec_()
    query.next()
    return
Patient(query.value(0),query.value(1),query.value(2),query.value(3),query.value(4),query.value(5),query.value(6),
```

```
query.value(7),query.value(8),query.value(9),query.value(10),query.value(11),query.value(12),query.value(13),qu
ery.value(14),query.value(15),query.value(16),query.value(17))

def createPatient():
    """ Returns a new database patient.
    """
    # Create an patient with no name and a new unique ID
    query = db.Query()
    query.prepare("INSERT INTO pacientes(Fecha_de_insercion) VALUES(CURRENT_DATE())")
    print(query.exec_())
    return Patient(query.lastInsertId(), '', '', '', '', '', '', '', '', '', '', '', '', '', '', '')

def removePatient(id):
    """ Removes a database patient having <id> in the database.
    """
    query = db.Query()
    query.prepare("DELETE FROM pacientes WHERE id =?")
    query.addBindValue(id)
    query.exec_()
```

labo.py

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import db

class Labo(object):
    """ This class contains the attributes and methods that can be applied on a
        laboratory.
    """
    def __init__(self, id, Nombre, Activo):
        """ The constructor initializes the attributes of a laboratory with the
            values given as parameters.
        """
        self._id = id
        self._Nombre = Nombre
        self._Activo = Activo

    """ The following methods are used to get the attributes of a laboratory.
    """
    def id(self):
        return self._id

    def nombre(self):
        return self._Nombre

    def activo(self):
        return self._Activo

    """ The following methods are used to set <value> as the value of an
        attribute of a laboratory.
    """
    def setId(self,value):
        self._id = value

    def setNombre(self,value):
        self._Nombre = value

    def setActivo(self,value):
        self._Activo = value

    def save(self, option):
        """ Updates, in the database, the attributes of a database laboratory
            added or modified by the user in labodialog. Remind that, when adding
            a laboratory, it is saved in the database with empty attributes by
            default.
```

```
        """
        query = db.Query()
        query.prepare("UPDATE laboratorios SET Nombre = ?, Activo =? WHERE id = ?")
        query.addBindValue(self.nombre())
        query.addBindValue(self.activo())
        query.addBindValue(self.id())
        query.exec_()

def getLabo(id):
    """ Returns the database laboratory corresponding to the given <id>.
    """
    query = db.Query()
    query.prepare("SELECT id, Nombre, Activo FROM laboratorios WHERE id =?")
    query.addBindValue(id)
    query.exec_()
    query.next()
    return Labo(query.value(0),query.value(1), query.value(2))

def createLabo():
    """ Returns a new database laboratory.
    """
    query = db.Query()
    query.prepare("INSERT INTO laboratorios (Nombre) VALUES(\"\")")
    query.exec_()
    return Labo(query.lastInsertId(),'','')

def removeLabo(id):
    """ Removes a database laboratory having <id> in the database.
    """
    query = db.Query()
    query.prepare("DELETE FROM laboratorios WHERE id=?")
    query.addBindValue(id)
    query.exec_()

def allLabos(entry):
    """ Returns a list with all the laboratories from the database including
        <entry> in their name. This parameter can be empty.
    """
    query = db.Query()
    query.prepare("SELECT id, Nombre, Activo FROM laboratorios WHERE Nombre LIKE?")
    query.addBindValue('%' + entry + '%')
    query.exec_()
    labos = []
    while query.next():
        labos.append(Labo(query.value(0),query.value(1),query.value(2)))
    return labos
```

test.py

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import db

class Test(object):
    """ This class contains the attributes and methods that can be applied on a
        test.
    """
    def __init__(self, id, Labo_id, Labo_nombre, Nombre, Tipo, Valores_posibles, Multiples_resultados, Activo):
        """ The constructor initializes the attributes of a test with the
            values given as parameters.
        """
        self._id = id
        self._Labo_id = Labo_id
        self._Labo_nombre = Labo_nombre
        self._Nombre = Nombre
```

```
        self._Tipo = Tipo
        self._Valores_posibles = Valores_posibles
        self._Multiples_resultados = Multiples_resultados
        self._Activo = Activo

    """ The following methods are used to get the attributes of a test.
    """
    def id(self):
        return self._id

    def labo_id(self):
        return self._Labo_id

    def labo_nombre(self):
        return self._Labo_nombre

    def nombre(self):
        nom = self._Nombre.replace('_',' ')
        return nom

    def tipo(self):
        return self._Tipo

    def valores_posibles(self):
        return self._Valores_posibles

    def multiples_resultados(self):
        return self._Multiples_resultados

    def activo(self):
        return self._Activo

    """ The following methods are used to set <value> as the value of an
        attribute of a test.
    """
    def setId(self,value):
        self._id = value

    def setLabo_id(self,value):
        self._Labo_id = value

    def setLabo_nombre(self,value):
        self._Labo_nombre = value

    def setNombre(self,value):
        self._Nombre = value

    def setTipo(self,value):
        self._Tipo = value

    def setValores_posibles(self,value):
        self._Valores_posibles = value

    def setMultiples_resultados(self, value):
        self._Multiples_resultados = value

    def setActivo(self, value):
        self._Activo = value

    def save(self, option):
        """ Updates, in the database, the attributes of a database test added or
            modified by the user in testdialog. Remind that, when adding a test,
            it is saved in the database with empty attributes by default.
        """
        query = db.Query()
        query.prepare("UPDATE examenes SET Labo_id = ?, Nombre = ?, Tipo =?, Valores_posibles = ?,
Multiples_resultados = ?, Activo = ? WHERE id = ?")
        query.addBindValue(self.labo_id())
```

```
        query.addBindValue(self.nombre())
        query.addBindValue(self.tipo())
        query.addBindValue(self.valores_posibles())
        query.addBindValue(self.multiples_resultados())
        query.addBindValue(self.activo())
        query.addBindValue(self.id())
        query.exec_()

def getTest(id):
    """ Returns the database test corresponding to the given <id>.
    """
    query = db.Query()
    query.prepare("SELECT examenes.id, examenes.Labo_id, laboratorios.Nombre, examenes.Nombre,
examenes.Tipo, examenes.Valores_posibles, examenes.Multiples_resultados, examenes.Activo FROM
examenes LEFT JOIN laboratorios ON laboratorios.id = examenes.Labo_id WHERE examenes.id =?")
    query.addBindValue(id)
    query.exec_()
    query.next()
    return Test(query.value(0),query.value(1), query.value(2), query.value(3), query.value(4), query.value(5),
query.value(6), query.value(7))

def createTest():
    """ Returns a new database test.
    """
    query = db.Query()
    query.prepare("INSERT INTO examenes (Nombre) VALUES(\"\")")
    query.exec_()
    return Test(query.lastInsertId(),'','','','','','')

def removeTest(id):
    """ Removes a database test having <id> in the database.
    """
    query = db.Query()
    query.prepare("DELETE FROM examenes WHERE id=?")
    query.addBindValue(id)
    query.exec_()

def allTests(entry):
    """ Returns a list with all the tests from the database including <entry> in
        their name. This parameter can be empty.
    """
    query = db.Query()
    query.prepare("SELECT examenes.id, examenes.Labo_id, laboratorios.Nombre, examenes.Nombre,
examenes.Tipo, examenes.Valores_posibles, examenes.Multiples_resultados, examenes.Activo FROM
examenes LEFT JOIN laboratorios ON laboratorios.id = examenes.Labo_id WHERE examenes.nombre LIKE ?")
    query.addBindValue('%' + entry + '%')
    print(query.exec_())
    tests = []
    while query.next():
        tests.append(Test(query.value(0),query.value(1),query.value(2), query.value(3), query.value(4),
query.value(5), query.value(6), query.value(7)))
    return tests
```

result.py

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import db

class Result(object):
    """ This class contains the attributes and methods that can be applied on a
        result.
    """
    def __init__(self,Labo_id, Examen_id, Nombre, Tipo, Valores_posibles, Multiples_resultados, Paciente_id,
Valor_text, Valor_date, Valor_float, Valor_int, Fecha_de_insercion):
```

```
        """ The constructor initializes the attributes of a result with the
            values given as parameters.
        """
        self._Labo_id = Labo_id
        self._Examen_id = Examen_id
        self._Nombre = Nombre
        self._Tipo = Tipo
        self._Valores_posibles = Valores_posibles
        self._Multiples_resultados = Multiples_resultados
        self._Paciente_id = Paciente_id
        self._Valor_text = Valor_text
        self._Valor_date = Valor_date
        self._Valor_float = Valor_float
        self._Valor_int = Valor_int
        self._Fecha_de_insercion = Fecha_de_insercion

    """ The following methods are used to get the attributes of a result.
    """
    def labo_id(self):
        return self._Labo_id

    def examen_id(self):
        return self._Examen_id

    def nombre(self):
        return self._Nombre

    def tipo(self):
        return self._Tipo

    def valores_posibles(self):
        return self._Valores_posibles

    def multiples_resultados(self):
        return self._Multiples_resultados

    def paciente_id(self):
        return self._Paciente_id

    def valor_text(self):
        return self._Valor_text

    def valor_date(self):
        return self._Valor_date

    def valor_float(self):
        return self._Valor_float

    def valor_int(self):
        return self._Valor_int

    def fecha_de_insercion(self):
        return self._Fecha_de_insercion

    """ The following methods are used to set <value> as the value of an
        attribute of a result.
    """
    def setLabo_id(self, value):
        self._Labo_id = value

    def setExamen_id(self, value):
        self._Examen_id = value

    def setNombre(self, value):
        self._Nombre = value

    def setTipo(self, value):
        self._Tipo = value
```

```
    def setValores_posibles(self, value):
        self._Valores_posibles = value

    def setMultiples_resultados(self, value):
        self._Multiples_resultados = value

    def setPaciente_id(self, value):
        self._Paciente_id = value

    def setValor_text(self, value):
        self._Valor_text = value

    def setValor_date(self, value):
        self._Valor_date = value

    def setValor_float(self, value):
        self._Valor_float= value

    def setValor_int(self, value):
        self._Valor_int = value

    def setFecha_de_insercion(self, value):
        self._Fecha_de_insercion = value

    def save(self, option):
        """ Saves or updates, in the database, the attributes of a database
            result added or modified by the user in resultsdialog.
        """
        if not self.multiples_resultados():
            # Only one result allowed for this test, remove all the previous ones
            query = db.Query()
            query.prepare("DELETE FROM resultados WHERE Paciente_id=? AND Labo_id=? AND Examen_id=?;")
            query.addBindValue(self.paciente_id())
            query.addBindValue(self.labo_id())
            query.addBindValue(self.examen_id())
            query.exec_()

        # Insert the new version of the result
        query = db.Query()
        query.prepare("INSERT INTO resultados(Paciente_id, Labo_id, Examen_id, Valor_text, Valor_date,
Valor_float, Valor_int, Fecha_de_insercion) VALUES (?,?,?,?,?,?,?,?);")
        query.addBindValue(self.paciente_id())
        query.addBindValue(self.labo_id())
        query.addBindValue(self.examen_id())
        query.addBindValue(self.valor_text())
        query.addBindValue(self.valor_date())
        query.addBindValue(self.valor_float())
        query.addBindValue(self.valor_int())
        query.addBindValue(self.fecha_de_insercion())
        query.exec_()

def allResults(Labo_id, Paciente_id):
    """ Returns a list with all the database results from the laboratory having
        <Labo_id> as id and about the patient having <Paciente_id> as id.
    """
    query = db.Query()
    query.prepare("SELECT examenes.Labo_id, examenes.id, examenes.Nombre, examenes.Tipo,
examenes.Valores_posibles, examenes.Multiples_resultados, resultados.Paciente_id, resultados.Valor_text,
resultados.Valor_date, resultados.Valor_float, resultados.Valor_int, resultados.Fecha_de_insercion FROM
examenes LEFT JOIN resultados ON resultados.Examen_id=examenes.id AND resultados.Paciente_id = ?
WHERE examenes.Labo_id = ? AND examenes.Activo = 1 ORDER BY examenes.id ASC,
resultados.Fecha_de_insercion ASC;")
    query.addBindValue(Paciente_id)
    query.addBindValue(Labo_id)
    query.exec_()
    results = []
```

```
    while query.next():

results.append(Result(query.value(0),query.value(1),query.value(2),query.value(3),query.value(4),query.value(5),
query.value(6),query.value(7),query.value(8),query.value(9),query.value(10),query.value(11)))

    return results
```

resultwidget.py

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

import copy
from result import *

class ResultWidget(QWidget):
    """ Displays the results of a medical test, ordered by date.

        The widget allows the user to choose for which date the results are to
        be displayed. Results can be edited and new results can be added by changing
        the ones of "today"
    """

    def __init__(self, parent):
        """ Create a new result widget
        """
        super().__init__(parent)

        # Create the layout
        self._layout = QHBoxLayout(self)
        self._layout.setContentsMargins(0, 0, 0, 0)

        # Create the combo box that will display the dates
        self._dates = QComboBox(self)
        self._layout.addWidget(self._dates)

        # We don't know yet the type of test that we have, so we cannot create the
        # widget that will display the results themselves
        self._widget = None
        self._results = []
        self._modified = False

        # When the user changes the date, display its corresponding result
        self._dates.currentIndexChanged.connect(self.dateChanged)

    def addResult(self, result):
        """ Add a result to this list of results
        """
        if self._widget is None:
            # This is the first result that we see, use it in order to create the
            # widget that corresponds to its test
            self._type = result.tipo()
            self._values = result.valores_posibles().split('|')
            self._multiple_values = result.multiples_resultados()
            self._first_result = result

            # Allow to choose a date only if multiple values are supported
            self._dates.setVisible(self._multiple_values)

            # Insert the widget before the combo box
            self._widget = self.createWidget()
            self._layout.insertWidget(0, self._widget)

        if result.paciente_id():
            # This result exists for this patient, add it to the list of results
```

```
            # we know
            self._results.append(result)
            self._dates.addItem(result.fecha_de_insercion().toString('dd/MM/yyyy'))
            self._dates.setCurrentIndex(self._dates.count() - 1)

    def dateChanged(self, index):
        """ Called when the user chooses a date in the combo box
        """
        self._modified = False

        if len(self._results) == 0:
            # No previous result to display, clear the widget
            self._clear_widget(self._widget)

        else:
            # Display the result selected. Selecting "today" displays the latest result
            self._set_value_of_widget(self._widget, self._get_result(self._results[index]))

    def result(self):
        """ Return an updated result (for the current date) if the user has changed
            it, otherwise return None.
        """
        if self._modified:
            # The user has changed the result: create a new one, with the date
            # of today and the updated result
            r = copy.copy(self._first_result)
            r.setFecha_de_insercion(QDate.currentDate())

            # Update the result with what the user has entered
            self._set_result(r, self._get_value_from_widget(self._widget))

            return r
        else:
            # No change, no need to save this result in the database
            return None

    def setModified(self):
        """ Called when the user changes the value of an input widget, so that
            the field is considered modified (and will be stored into the database)
        """
        self._modified = True

    def createWidget(self):
        """ Depending of the type of the exam and its result, this method
            is used to generate a widget for this exam and to set its result.

            It also defines which getter/setter to use in order to read and write
            the result (different widgets have different getters/setters, for
            instance setDate or setText, and Result can be accessed using
            valor_text, valor_int, etc)
        """
        # Date
        if self._type == 0:
            widget = QDateEdit(self)
            widget.setDisplayFormat('dd/MM/yyyy')
            widget.dateChanged.connect(self.setModified)

            self._get_value_from_widget = QDateEdit.date
            self._set_value_of_widget = QDateEdit.setDate

            # Clearing a date field is done by setting it to 01/01/2000
            self._clear_widget = lambda w: w.setDate(QDate(2000, 1, 1))
            self._get_result = Result.valor_date
            self._set_result = Result.setValor_date

        # Float
        elif self._type == 1:
            widget = QDoubleSpinBox(self)
```

```
            widget.setMinimum(-2000000000.)
            widget.setMaximum(2000000000.)
            widget.valueChanged.connect(self.setModified)

            self._get_value_from_widget = QDoubleSpinBox.value
            self._set_value_of_widget = QDoubleSpinBox.setValue
            # Clearing a double spinbox is done by setting it to 0
            self._clear_widget = lambda w: w.setValue(0.0)
            self._get_result = Result.valor_float
            self._set_result = Result.setValor_float
        # Int
        elif self._type == 2:
            widget = QSpinBox(self)
            widget.setMinimum(-2000000000)
            widget.setMaximum(2000000000)
            widget.valueChanged.connect(self.setModified)

            self._get_value_from_widget = QSpinBox.value
            self._set_value_of_widget = QSpinBox.setValue
            self._clear_widget = lambda w: w.setValue(0)
            self._get_result = Result.valor_int
            self._set_result = Result.setValor_int

        # Text
        elif self._type == 3:
            widget = QLineEdit(self)
            widget.textEdited.connect(self.setModified)

            self._get_value_from_widget = QLineEdit.text
            self._set_value_of_widget = QLineEdit.setText
            # Clearing a line edit is done by putting an empty string in it
            self._clear_widget = lambda w: w.setText('')
            self._get_result = Result.valor_text
            self._set_result = Result.setValor_text

        # ComboBox/List
        elif self._type == 4:
            widget = QComboBox(self)

            widget.addItem('')
            for value in self._values:
                widget.addItem(value)

            widget.currentIndexChanged.connect(self.setModified)

            self._get_value_from_widget = QComboBox.currentIndex
            self._set_value_of_widget = QComboBox.setCurrentIndex
            # Clearing a list is done by selecting its first element,
            # which for each one, it is empty
            self._clear_widget = lambda w: w.setCurrentIndex(0)
            self._get_result = Result.valor_int
            self._set_result = Result.setValor_int

        widget.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        return widget
```

Abstractlist et ses fichiers associés

abstractlist.py

```
from PyQt5.QtCore import *
```

```
from PyQt5.QtWidgets import *

from ui_abstractlist import *

class AbstractList(QWidget):
    """ AbstractList is the base class of all the lists in the program :
        UserList, PatientList, LaboList and TestList. This abstract class
        provides a list with columns, and items can be added, removed or edited
        using buttons.
    """
    def __init__(self, parent):
        """ The constructor creates three buttons "Remove", "Add..." and
            "Edit..." that will appear in the classes that inherit this abstract
            class.
        """
        QWidget.__init__(self, parent)
        self.ui = Ui_AbstractList()
        self.ui.setupUi(self)
        self.ui.btnRemove.setVisible(self.removeEnabled())
        self.ui.btnRemove.clicked.connect(self.removeClicked)
        self.ui.btnAdd.clicked.connect(self.addClicked)
        self.ui.btnEdit.clicked.connect(self.editClicked)
        self.ui.btnBuscar.clicked.connect(self.refresh)
        self.ui.txtBuscar.returnPressed.connect(self.ui.btnBuscar.click)
        self.ui.btnActualizar.clicked.connect(self.refreshAll)

        # Create the columns in the list
        self.ui.list.setHeaderLabels(self.columns())

        # Load the list when the program starts, as the database can contain
        # entries before the user has clicked any button
        self.populate("")

    def columns(self):
        """ Returns a list of strings, one string per column in the list.
        """
        pass

    def populate(self):
        """ Fills the list with data from the database.
        """
        pass

    def remove(self, id):
        """ Removes an element having <id> in the database.
        """
        pass

    def createDialog(self):
        """ Instantiates the edit/add dialog for this list.
        """
        pass

    def createNewObject(self):
        """ Returns a new database object, for instance a new user
            (user.createUser).
        """
        pass

    def getObject(self, id):
        """ Returns the database object corresponding to the given <id>.
        """
        pass

    def addItem(self, id, values):
        """ Adds a line to the list with id <id>

            This method takes as parameter <values> which is a list of the
```

```
            attributes of an object (patient, user, labo or test). Those
            attributes are entered on a line of the general list, in the
            adequate column.

        """
        # Create an empty entry in the list of entry
        entry = QTreeWidgetItem(self.ui.list)

        for i in range(len(values)):
            # fill the columns of the entry-line
            entry.setText(i, str(values[i]))

        # Tells to the line which is his corresponding ID
        entry.setData(0, Qt.UserRole, id)

    def refresh(self):
        """ Clears all the lines (corresponding to an object) of the list and
            then, fills the list with data from the database that you have
            filtered due to the search window.
        """
        self.ui.list.clear()
        self.populate(self.ui.txtBuscar.text())

    def refreshAll(self):
        """ Clears all the lines (corresponding to an object) of the list and
            then, fills the list with all the data from the database.
        """
        self.ui.list.clear()
        self.populate("")

    def removeClicked(self):
        """ Defines what happens when clicking on the button "Remove".

            This method takes the id of the object corresponding to the last
            line on which the user has clicked and removes the element having
            this id in the database. Then, the list is refreshed.
        """
        # This method returns the current element, the one clicked by the user
        currentItem = self.ui.list.currentItem()

        if currentItem is not None:
            id = currentItem.data(0, Qt.UserRole)
            self.remove(id)

        # We have removed an object, so update the list
        self.refresh()

    def addClicked(self):
        """ Defines what happens when clicking on the button "Add...".

            This method shows a dialog box. The user fills it with the
            attributes of the new object that he wants to create. It creates
            a new database object, those attributes are associated to him and
            the object is saved in the database. Then, the list is refreshed.
        """
        # Show the dialog. exec_() when the user clicks on Ok/Cancel
        dlg = self.createDialog()


        #Show the dialog as a modal dialog, blocking until the user closes it.
        if dlg.exec_() == QDialog.Accepted:
            newObject = self.createNewObject()

            # Ask the dialog to populate the new object
            dlg.updateObject(newObject)

            # Now that newUser has valid information, save it in the database
            newObject.save(1)
```

```
        # We have added an object, so update the list
        self.refresh()

    def editClicked(self):
        """ Defines what happens when clicking on the button "Edit...".

            This method opens a dialog box filled with the attributes of the
            database object corresponding to the last line of the list on which
            the user has clicked. The user can modify those attributes. The new
            attributes are associated to the object. This object is saved in the
            database and then, the list is refreshed.
        """
        currentItem = self.ui.list.currentItem()

        if currentItem is not None:
            id = currentItem.data(0, Qt.UserRole)
            obj = self.getObject(id)

            # Open UserDialog and populate it with the current object
            dlg = self.createDialog()
            dlg.displayObject(obj)

            # Open the dialog, and if the user clicks on Ok, update the user in
            # the database
            if dlg.exec_() == QDialog.Accepted:
                # Copy what has been modified in the dialog back into the object
                dlg.updateObject(obj)

                # And save the object
                obj.save(1)

        self.refresh()

    def removeEnabled(self):
        """ Returns the status of the button 'Remove'. In the tabs
            laboratrios and examenes it does not appear
        """
        return True
```

userlist.py

```
from PyQt5.QtWidgets import *
from abstractlist import *
from userdialog import *
import user
import id

class UserList(AbstractList):
    """ UserList is a derived class of the base class AbstractList. It
        provides a list of user, with columns. The attributes of each
        user are written on each line of the list, in the adequate column.
        Other users can be added, removed or edited using buttons.
    """

    def __init__(self, parent):
        """ The constructor calls the constructor of the base class :
            AbstractList.
        """
        AbstractList.__init__(self, parent)

    def columns(self):
        """ Returns a list of strings, one string per column in the list.
        """
```

```
        return ['Apellido', 'Nombre', 'Cédula de identidad','Nacimiento', 'Teléfono','Email','Domicilio']

    def populate(self, entry):
        """ Fills the list with users from the database including <entry> in
            their name, first name or id. This parameter can be empty.
        """
        ll = user.allUsers(entry)
        for u in ll:
            if u.ID() == id.current_id():
                pass
            else:
                # Each item is composed by the ID and a list containing all
                # the informations related to the user
                self.addItem(u.ID(), [u.apellido(), u.nombre(),
u.CI(),u.nacimiento().toString(Qt.SystemLocaleShortDate), u.telefono(), u.email(), u.domicilio()])

    def remove(self, id):
        """ Removes the user of the correspoding ID
        """
        user.removeUser(id)

    def createDialog(self):
        """ Inizialises the edit/add dialog for this list
            """
        return UserDialog(self)

    def createNewObject(self):
        """ Returns a new user, that for the moment will just had the ID as
        information
        """
        return user.createUser()

    def getObject(self, id):
        """ Returns the database patient corresponding to the given ID
        """
        return user.getUser(id)
```

patientlist.py

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

from abstractlist import *
from patientdialog import *
import patient

class PatientList(AbstractList):
    """ PatientList is a derived class of the base class AbstractList. It
        provides a list of patients, with columns. The attributes of each
        patient are written on each line of the list, in the adequate column.
        Other patients can be added, removed or edited using buttons.
    """
    def __init__(self, parent):
        """ The constructor calls the constructor of the base class :
            AbstractList.
        """
        AbstractList.__init__(self, parent)

    def columns(self):
        """ Returns a list of strings, one string per column in the list.
        """
        return ['ID','Fecha consulta', 'Historia Clinica', 'Nombre', 'Apellido', 'Carnet de Identidad','Fecha de
nacimiento', 'Estado civil', 'Peso', 'Talla', 'Establecimiento o centro de salud', 'Provincia', 'Municipio', 'Localidad',
'Direccion', 'Numero de telefono 1', 'Numero de telefono 2', 'Croquis vivienda']
```

```
    def populate(self,entry):
        """ Fills the list with patients from the database including <entry> in
            their name, first name or id. This parameter can be empty.
        """
        estados_civil = ['','Casada', 'Soltera', 'Divorciada', 'Viuda', 'Otros']
        croquis_vivienda = ['','Si', 'No']
        for p in patient.allPatients(entry):
            self.addItem(p.id(), [
                p.id(),
                p.fecha_consulta().toString(Qt.SystemLocaleShortDate),
                p.h_Clinica(),
                p.nombre(),
                p.apellido(),
                p.carnet_de_identidad(),
                p.fecha_de_nacimiento().toString(Qt.SystemLocaleShortDate),
                estados_civil[p.estado_civil()],
                p.peso(),
                p.talla(),
                p.establecimiento_o_centro_de_salud(),
                p.provincia(),
                p.municipio(),
                p.localidad(),
                p.direccion(),
                p.numero_de_telefono_1(),
                p.numero_de_telefono_2(),
                croquis_vivienda[p.croquis_vivienda()]
            ])

    def remove(self, id):
        """ Removes a patient having <id> in the database.
        """
        patient.removePatient(id)

    def createDialog(self):
        """ Instantiates the edit/add dialog for this list.
        """
        return PatientDialog(self)

    def createNewObject(self):
        """ Returns a new database patient.
        """
        return patient.createPatient()

    def getObject(self, id):
        """ Returns the database patient corresponding to the given <id>.
        """
        return patient.getPatient(id)
```

labolist.py

```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *

from abstractlist import *
from labodialog import *
import labo
import id

class LaboList(AbstractList):
    """ LaboList is a derived class of the base class AbstractList. It provides
        a list of laboratories, with a column. The name of each laboratory is
        written on each line of the list. Other laboratories can be added,
        removed or edited using buttons.
    """
    def __init__(self, parent):
```

```
        """ The constructor calls the constructor of the base class :
            AbstractList.
        """
        AbstractList.__init__(self, parent)

    def columns(self):
        """ Returns a string corresponding to the column of the list.
        """
        return ['Nombre']

    def populate(self, entry):
        """ Fills the list with laboratories from the database, only if the
            laboratories can be modified by the user and if their name
            includes <entry>. This parameter <entry> can be empty.
        """
        usr = id.current_user()

        for l in labo.allLabos(entry):
            if l.id() in usr.allowedLabosModifs():
                self.addItem(l.id(), [l.nombre().replace('_',' ')])

    def remove(self, id):
        """ Removes a laboratory having <id> in the database.
        """
        labo.removeLabo(id)

    def createDialog(self):
        """ Instantiates the edit/add dialog for this list.
        """
        return LaboDialog(self)

    def createNewObject(self):
        """ Returns a new database laboratory.
        """
        return labo.createLabo()

    def getObject(self,id):
        """ Returns the database laboratory corresponding to the given <id>.
        """
        return labo.getLabo(id)

    def removeEnabled(self):
        """ Returns the status of the button "Remove". This button does not
            appear in LaboList.
        """
        return False
```

testlist.py

```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *

from abstractlist import *
from testdialog import *
import test
import id

class TestList(AbstractList):
    """ TestList is a derived class of the base class AbstractList. It provides
        a list of tests, with columns. The attributes of each test are written
        on each line of the list, in the adequate column. Other tests can be
        added, removed or edited using buttons.
    """
    def __init__(self, parent):
        """ The constructor calls the constructor of the base class :
```

```
            AbstractList.
        """
        AbstractList.__init__(self, parent)

    def columns(self):
        """ Returns a list of strings, one string per column in the list.
        """
        return ['Labo nombre', 'Nombre', 'Tipo', 'Valores posibles']

    def populate(self,entry):
        """ Fills the list with tests from the database, only if the laboratory
            of the tests can be modified by the user and if their name includes
            <entry>. This parameter <entry> can be empty.
        """
        test_types = ['Fecha', 'Numero con decimales', 'Numero entero', 'Texto', 'Multiopción']
        usr = id.current_user()

        for t in test.allTests(entry):

            # The labo of the test is in the allowed labo list of the user
            if t.labo_id() in usr.allowedLabosModifs():
                self.addItem(t.id(), [t.labo_nombre().replace('_',' '),t.nombre().replace('_',' '),test_types[t.tipo()],t.valores_posibles()])

    def remove(self, id):
        """ Removes a test having <id> in the database.
        """
        test.removeTest(id)

    def createDialog(self):
        """ Instantiates the edit/add dialog for this list.
        """
        return TestDialog(self)

    def createNewObject(self):
        """ Returns a new database test.
        """
        return test.createTest()

    def getObject(self,id):
        """ Returns the database test corresponding to the given <id>.
        """
        return test.getTest(id)

    def removeEnabled(self):
        """ Returns the status of the button "Remove". This button does not
            appear in TestList.
        """
        return False
```

Fichiers de support

main.py

```
from PyQt5.QtWidgets import *
import db
from logindialog import *
from mainwindow import *
import user

def main():
    """ Initializes the application and shows the interface. Fisrt logindialog
    who demands the user to identificate himself, in case of authentication,
    displays the main window, with the tabs corresponding to the priviliges of
```

```
    the user
    """
    app = QApplication([])

    loginDialog = LoginDialog()

    while True:
        if loginDialog.exec_() == QDialog.Rejected:
            return

        # Check that the password that the user has entered is correct
        correct, ID = user.login(loginDialog.CI(), loginDialog.password())
        if correct:
            # Ok, connected
            id.set_id(ID)
            break
        else:
            # Bad username or password, tell the user to try again
            QMessageBox.critical(None,
                                 "Acceso denegado",
                                 "Cédula de identidad y/o contraseña incorrectos")

    # The user has provided the correct password, show the main window
    usr = id.current_user()
    win = MainWindow(ID)

    # ss is the lisr of the grups with the privilege of managing users
    # and j contains the ones ables tp modife the databse entries
    ss = [1,2,3,4,5,6,7,8]
    j = [1,2,3,4,9,10,11,12]
    i = 1
    if usr.grupo() not in ss:
        win.notAdmin()
        i = 0
    else:
        pass

    if usr.grupo() not in j:
        win.notModif(i)
    else:
        pass

    win.show()

    # Let the user interact with the main window before closing the application
    app.exec_()

main()
```

exportexcell.py

```
#!/bin/python
from PyQt5.QtCore import *
from PyQt5.QtWidgets import*
import db

def exportcsv(extra_select, extra_select_names, extra_conditions, bind_values):
    """ Dump the patient list to a CSV file (that can be opened in Excel). Personal
        and medical data about the patients are written to the file.

        extra_select: additional columns to select (p.Nombre for instance)
        extra_select_names: names associated with the columns selected (Nombre for instance)
        extra_conditions: code that can be inserted in the SQL query in the WHERE
                    clause, to further filter the data (filter on dates or
                    exam types for instance)
```

```
    bind_values: list of values to bind to the query using addBindValue. This
                 can be used if extra_conditions contains question marks.
    """
    query = db.Query()
    query.prepare("""
    SELECT
        r.Valor_text,
        r.Valor_date,
        r.Valor_float,
        r.Valor_int,
        e.id,
        e.Nombre,
        e.Tipo,
        e.Valores_posibles,
        l.Nombre,

        %s
    FROM
        examenes e
    LEFT JOIN resultados r ON r.Examen_id=e.id
    LEFT JOIN pacientes p ON p.id=r.Paciente_id
    LEFT JOIN laboratorios l ON l.id=r.Labo_id AND l.Activo=1
    WHERE
        e.Activo=1
        %s
    ORDER BY
        p.id DESC
    """ % (extra_select, extra_conditions))

    for data in bind_values:
        query.addBindValue(data)

    query.exec_()

    patients = []
    labos = {'': extra_select_names}

    last_patient_id = None
    last_patient = None

    while query.next():

        # Create a new patient if we change the patient
        if query.value(9) != last_patient_id:
            if last_patient is not None:

                # A patient is finished, add it to the list
                patients.append(last_patient)

            # Create a new patient
            last_patient_id = query.value(9)
            last_patient = {}

            # Personal informations
            for i in range(len(extra_select_names)):
                v = query.value(9 + i)

                # Put the dates in the correct format
                if isinstance(v, QDate):
                    v = v.toString('dd/MM/yyyy')

                last_patient['', extra_select_names[i])] = str(v)

        # Column of the result
        labo = query.value(8)
        test = query.value(5)

        if labo not in labos:
```

```
            labos[labo] = []

        if test not in labos[labo]:
            labos[labo].append(test)

        # Value of the result
        result = ''
        tipo = int(query.value(6))
        valores_posibles = str(query.value(7)).split('|')

        if tipo == 0:
            result = query.value(1).toString('dd/MM/yyyy')
        elif tipo == 1:
            result = query.value(2)
        elif tipo == 2:
            result = query.value(3)
        elif tipo == 3:
            result = query.value(0)
        elif tipo == 4:
            try:
                result = valores_posibles[query.value(3)]
            except:
                result = ''

        # Stock the result
        last_patient[(labo, test)] = str(result)

    patients.append(last_patient)

    # Generate the CSV file
    filename = QFileDialog.getSaveFileName(None,"Save File",
QStandardPaths.writableLocation(QStandardPaths.DocumentsLocation),"CSV files (*.csv)")
    fichier = open(filename[0], 'w+')

    for labo, columns in labos.items():
        print(labo + (';' * len(columns)), end='', file = fichier)

    print('', file = fichier)

    for labo, columns in labos.items():

        # The method join() returns a string in which the string elements of sequence have been joined by str
separator.
        print(';'.join(columns) + ';', end='', file = fichier)

    print('', file = fichier)

    for patient in patients:
        for labo, columns in labos.items():
            for test in columns:
                print(patient.get((labo, test), '') + ';', end='', file = fichier)

        print('', file = fichier)
    fichier.close()

def exportexcel(date1, date2):
    """ Creates the csv file that will contain the columes that sumarizes the
        personal and medical data of the patients who have been inserted or
        modified during the period specified in the interface
    """

    exportcsv(
    """
        p.id,
        p.Nombre,
        p.Apellido,
        p.Numero_de_telefono_1,
        p.Numero_de_telefono_2
```

```
    """,
    ['ID', 'Nombre', 'Apellido', 'Teléfono1', 'Teléfono2'],
    """
        AND e.id IN (5, 77, 80, 81, 84, 36, 57, 61, 68, 69, 70)
        AND (
            p.Fecha_de_insercion BETWEEN ? AND ?
            OR p.id IN (
                SELECT DISTINCT Paciente_id
                FROM resultados
                WHERE (
                    Examen_id IN (5, 77, 80, 81, 84, 36, 57, 61, 68, 69, 70)
                    AND (Fecha_de_insercion BETWEEN ? AND ?)
                )
            )
        )
    """,
    [date1, date2, date1, date2])

def exportstats():
    """ Creates a csv file that will contain the medical information required
        by professor Veronique Fontaine to do her statistics
    """

    exportcsv(
    """
        p.id,
        p.Fecha_de_nacimiento,
        p.Estado_civil,
        p.Establecimiento_o_centro_de_salud,
        p.Provincia,
        p.Municipio,
        p.Localidad
    """,
    ['ID', 'Fecha de Nacimiento', 'Estado civil', 'Establecimiento o centro de salud','Provincia paciente','Municipio
paciente','Localidad paciente'],
    '',
    [])
```

id.py

```
""" This file is used to keep track of the ID of the current user of the database
"""
import user


user_id = None
user_instance = None

def set_id(Id):
    """ Convert a local variable (the id) to a global variable which will be callable
    """
    global user_id
    global user_instance

    user_id = Id
    user_instance = user.getUser(Id)

def current_id():
    """ Return the global variable id of the current user
    """
    return user_id

def current_user():
    """ Returns the current user as a global variable
    """
```

```
    return user_instance
```

db.py

```
from PyQt5.QtCore import *
import pymysql
from pymysql.constants import FIELD_TYPE

import datetime

# Connection to the database, None before the user enters his/her password
con = None

class Query(object):
    """ Query on the database, is used exactly like a db.Query
    """

    def __init__(self):
        if not con:
            raise ConnectionError('Database not open, please call db.connect(username, password)')
        else:
            self._cursor = con.cursor()

    def prepare(self, query):
        """ Prepare a query containing "?" marks when values will be placed
        """
        self._query = query.replace('?', '%s')
        self._args = []
        self._row = None

    def addBindValue(self, value):
        """ Add a parameter to the query being prepared
        """
        if isinstance(value, QDate):
            # Conver a QDate to a datetime.date used by PyMySQL
            value = datetime.date(value.year(), value.month(), value.day())

        self._args.append(value)

    def exec_(self, *args):
        """ Execute the last prepared query (if no args given), or directly execute
            the query passed as parameter
        """
        if len(args) == 1:
            # A query has been given as parameter
            self.prepare(args[0])

        # Execute the last prepared query
        try:
            self._cursor.execute(self._query, self._args)
        except pymysql.err.ProgrammingError as e:
            print('Database error: ' + str(e))
            print('In query: ' + self._query)

    def next(self):
        """ Advance one position in the results returned by the last executed query
        """
        self._row = self._cursor.fetchone()

        return self._row is not None

    def value(self, index):
        """ Get the value of one column of the current row
        """
        if not self._row:
```

```
            raise OverflowError('The MySQL query is not positioned on a valid record')

        val = self._row[index]

        if isinstance(val, datetime.date):
            # Convert a datetime.date to a QDate, used by the rest of the
            # project
            val = QDate(val.year, val.month, val.day)
        elif val is None:
            # Provide a default value for the field instead of None
            field = con._result.fields[index]

            if field.type_code in (FIELD_TYPE.DECIMAL, FIELD_TYPE.TINY, FIELD_TYPE.SHORT,
FIELD_TYPE.LONG, FIELD_TYPE.LONGLONG):
                # Integer, default to 0
                val = 0
            elif field.type_code in (FIELD_TYPE.FLOAT, FIELD_TYPE.DOUBLE):
                # Float, default to 0.0
                val = 0.0
            elif field.type_code in (FIELD_TYPE.VAR_STRING, FIELD_TYPE.STRING, FIELD_TYPE.BLOB):
                # String, default to the empty string
                val = ''
            elif field.type_code in (FIELD_TYPE.DATE,):
                # Date, default to 1/1/2000
                val = QDate(2000, 1, 1)

        return val

    def lastInsertId(self):
        """ Return the ID of the last inserted object
        """
        return self._cursor.lastrowid

def connect(username, password):
    """ Connect to the database by using the provided username and password.

        The server is the one of Codepo, and the database is "codepo"
    """
    global con

    con = pymysql.connect(
        host='151.80.132.131',
        user=username,
        password=password,
        db='codepo',
        # Enable the encryption, the .pem file contains the certificate of the
        # MySQL server so that the client (this program) can verify that it
        # is talking to the correct server
        ssl={
            'ca': 'mysql_ca_cert.pem'
        },
        # Any modification of the database (UPDATE, INSERT, etc) is immediately
        # visible to the other users
        autocommit=True)
```

setup.py

```
 from cx_Freeze import setup, Executable
import sys

""" This file complies and creates the folder with an executable, that will
allow to use the database without having installed packages like qt5,
mysql, python3,...
"""
#For Windows
```

```
if sys.platform == 'win32':
    base = 'Win32GUI'
    targetName = 'codepo.exe'
#For Linux
else:
    base = None
    targetName = 'codepo'

executables = [
    Executable('main.py', base=base, targetName=targetName)
]

setup(name='Codepo',
      version = '1.0',
      description = 'Codepo',
      executables = executables,
      options = {'build_exe': {
          'include_files': [('mysql_ca_cert.pem', '')],
      }, "bdist_mac": {
          "qt_menu_nib": "/opt/local/share/qt5/plugins/platforms",
      }})
```

Annexe D : Le manuel utilisateur (version française)

Projet Codepo

Création d'une base de données chargée du suivi médical de patientes atteintes du cancer du col de l'utérus en Bolivie

Table des matières

Manuel utilisateur

La base de données s'est voulue la plus intuitive possible et ses interfaces ont été conçues dans cette optique. Les fenêtres sont épurées de manière à aller à l'essentiel et leur design est redondant pour en faciliter la compréhension.

Ce manuel utilisateur a pour but de familiariser le personnel médical à l'utilisation de cette base de données. Pour ce faire, les différentes fenêtres d'interface seront passées en revue dans les détails, afin d'en décrire les principes et fonctionnement.

1. Comment lancer la base de données ?

Une fois le dossier « Build.zip » enregistré sur le bureau de l'utilisateur de la base de données, la première étape va consister à décompresser ce dossier. Pour ce faire, un clic droit sur le dossier permet d'afficher le menu déroulant où la mention « extraire » apparaît. En cliquant sur cette dernière, les fichiers seront décompressés dans un dossier équivalent. Une fois ce nouveau dossier ouvert, un simple double-clic sur le « codepo.exe » encadré ci-dessous (cf. Figure 1) permet d'arriver sur la première fenêtre d'interface de la base de données.

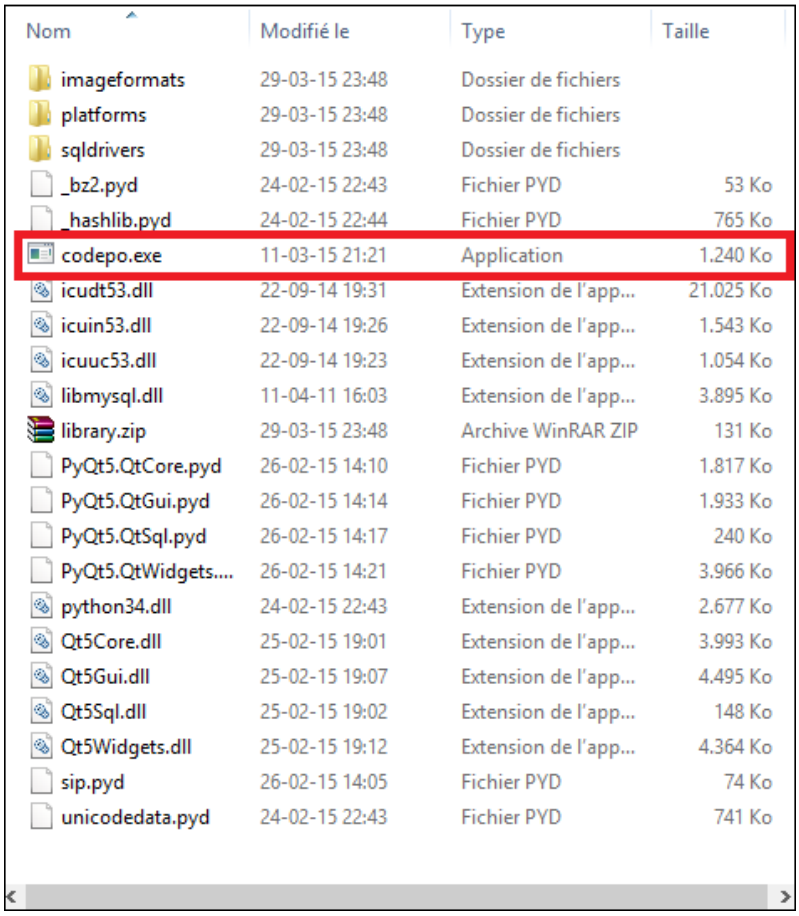


Figure 1

2. Fenêtre « Login »

Une fois le fichier « codepo.exe » lancé, la fenêtre d'identification s'affiche (cf. Figure 2). Dans cette fenêtre, entrer un identifiant et un mot de passe et cliquer sur le bouton « Ok » permet de se connecter à la base de données des patientes. La fenêtre du menu principal (cf. Figure 3) s'ouvre alors et l'utilisateur peut avoir accès aux données des patientes ou encore, ajouter des informations sur celles-ci. Il est important de rappeler que l'identifiant d'un utilisateur n'est autre que son numéro de registre national et que son mot de passe est initialement transmis par l'administrateur de la base de données.

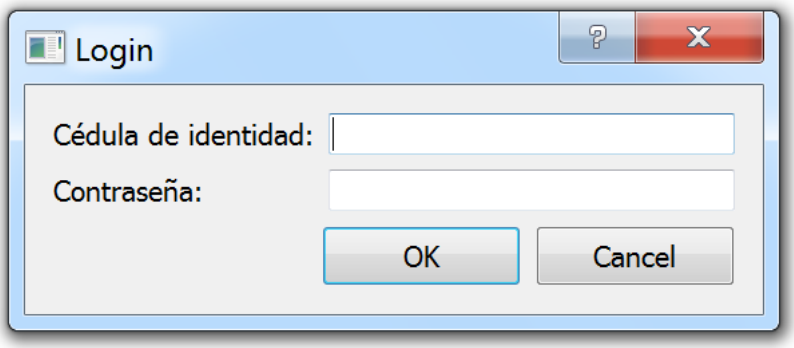


Figure 2

3. Fenêtre « Menu principal »

Après s'être connecté à la base de données, l'utilisateur arrive sur la fenêtre principale représentée à la Figure 4. Cette fenêtre joue un rôle central puisqu'elle donne accès à la plupart des applications possibles.

Tout d'abord, cette fenêtre dispose de deux boutons dans le haut. Celui de gauche « Completar datos personales » permet d'accéder à une nouvelle fenêtre d'interface dans laquelle il est possible de compléter ou de modifier les informations personnelles relatives à l'utilisateur.

Lors de sa première utilisation, il est demandé à chaque membre du personnel de compléter ses informations personnelles. Une fois le bouton « Completar datos personales » enfoncé, le formulaire visible à la Figure 3 apparaît et doit être complété. Les informations basiques sont requises mais, en outre, l'utilisateur a la possibilité de modifier son mot de passe. Il suffit d'encoder le nouveau dans l'emplacement à coté de « Cambiar contraseña ».

Dans le cas où un utilisateur encoderait un nouveau mot de passe en y glissant par inadvertance une erreur, celui-ci ne réussirait plus à se connecter à de données par la suite. C'est pourquoi un dispositif de vérification a été mis en place. L'utilisateur devra encoder deux fois son mot de passe et la modification ne se fera que dans le cas où ces deux champs sont égaux.

Il est à noter que l'utilisateur devra penser à sauvegarder les modifications amenées à ses données personnelles, avant de fermer cette fenêtre. Celui-ci pourra revenir changer à sa guise ses informations personnelles.

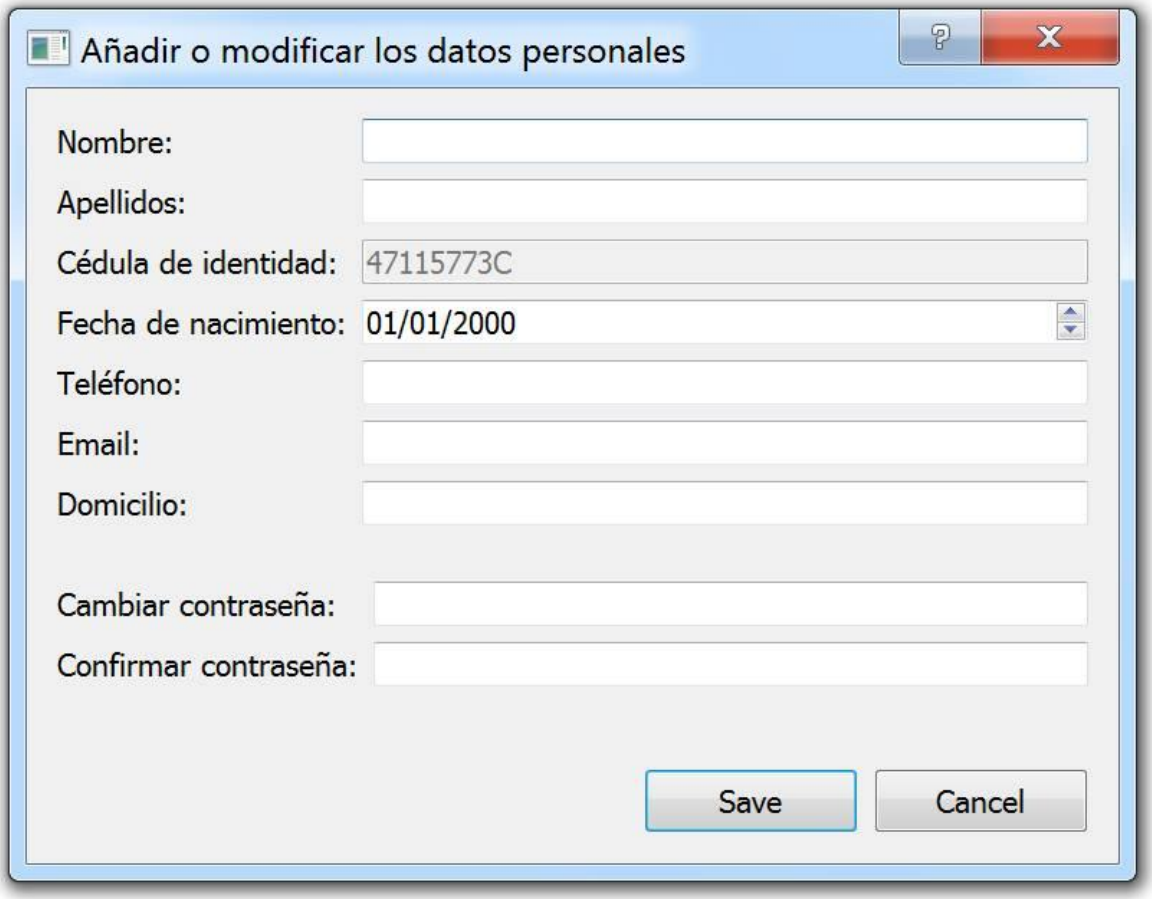


Figure 3

Le bouton de droite « Desconectarse de la base de datos » permet, quant à lui, de se déconnecter de la base de données. Afin de respecter la confidentialité des données sensibles figurant dans la base de données, il est demandé à chaque membre du personnel de se déconnecter après toute utilisation.

Ensuite, différents onglets sont disponibles au sein de ce « Menu principal ». L'utilisateur n'aura qu'à cliquer dessus afin d'accéder à leur contenu. Ces différents onglets seront explicités dans la suite de ce manuel, mais une brève explication est donnée ici :

- « Usuarios » est une liste des différentes personnes ayant accès à la base de données couplées à leurs informations personnelles;
- « Pacientes » constitue une liste des différentes patientes encodées dans la base ainsi que les données personnelles associées;
- « Laboratorios » est une liste des différents centres d'examens présents sur le site de l'UMSS ;
- « Entradas » reprend tous les tests effectués dans les différents laboratoires ;
- « Excel » permet d'extraire des données de la base avant de les afficher sous forme de tableau Excel.

3.1 Onglet « Excel »

Dans cette première fenêtre « Menu principal », l'onglet « Excel » est sélectionné (cf. Figure 4). Celui-ci propose deux actions à l'utilisateur.

La première consiste à générer un tableau Excel comprenant les résultats des patientes à certains tests sélectionnés au préalable par les soins de Véronique Fontaine. Ce fichier Excel sera utilisé dans le but d'effectuer des statistiques concernant le traitement des patientes du cancer du col de l'utérus. Celles-ci permettront notamment de vérifier que toutes les mesures pilotes mises en place dans la province de Cochabamba ces dernières années tendent à faire diminuer le taux de mortalité des patientes. Pour générer ce fichier Excel, l'utilisateur n'a qu'à enfoncer le bouton « Estadisticas ».

La deuxième action proposée permet d'exporter certaines informations cruciales des fichiers médicaux de patientes actuellement suivies vers un tableau Excel, afin d'en permettre l'impression. L'utilisateur doit encoder deux dates correspondant aux dernières modifications des informations relatives à une patiente entre lesquelles il souhaite effectuer sa recherche et ensuite, cliquer sur « Lista Pacientes ».

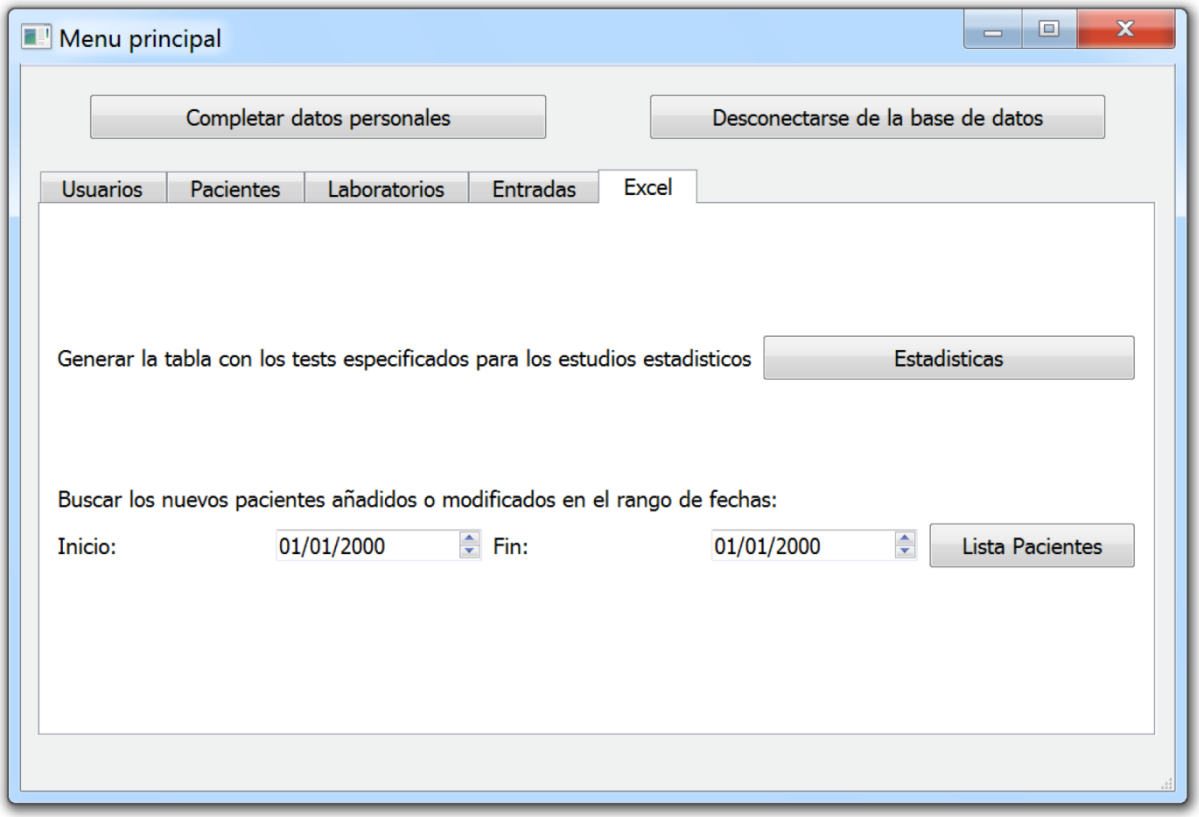


Figure 4

Une fois un de ces deux boutons enfoncés, une seconde fenêtre apparait, dans laquelle il est demandé à l'utilisateur de créer et enregistrer un fichier de type « CSV » au sein du dossier

« Documents » de son ordinateur. Une fois cet emplacement sélectionné, le fichier y restera accessible même après s'être déconnecté de la base de données.

3.2 Onglet « Usuarios »

Cet onglet, uniquement visible par les administrateurs de la base de données, offre une liste des différents utilisateurs de cette base, ainsi que leurs données personnelles. Une barre de recherche située dans le haut de cet onglet permet de taper le nom, prénom ou numéro de registre national d'un utilisateur avant de cliquer sur le bouton « Buscar », ce qui permet de lancer une recherche sur la séquence de lettres encodées. Seuls les utilisateurs dont le nom, prénom ou numéro de registre national comprend cette séquence persistent dans la liste d'utilisateurs affichée. Le bouton « Mostrar Todos » permet, quant à lui, de rafraîchir cette liste, afin de revenir à la liste complète du personnel médical utilisateur de la base de données.

Les administrateurs ont la possibilité de modifier cette liste d'utilisateurs via les trois boutons présents dans le bas de l'onglet.

Le premier (« Eliminar seleccionado ») permet de supprimer un utilisateur préalablement sélectionné dans la liste. L'ajout d'un nouvel utilisateur se fait via le bouton « Añadir… », mais il est également possible de modifier les données ou privilèges d'un utilisateur déjà encodé. Pour ce faire, il suffit de sélectionner la ligne correspondant à cette personne avant de cliquer sur le bouton « Editar… ».

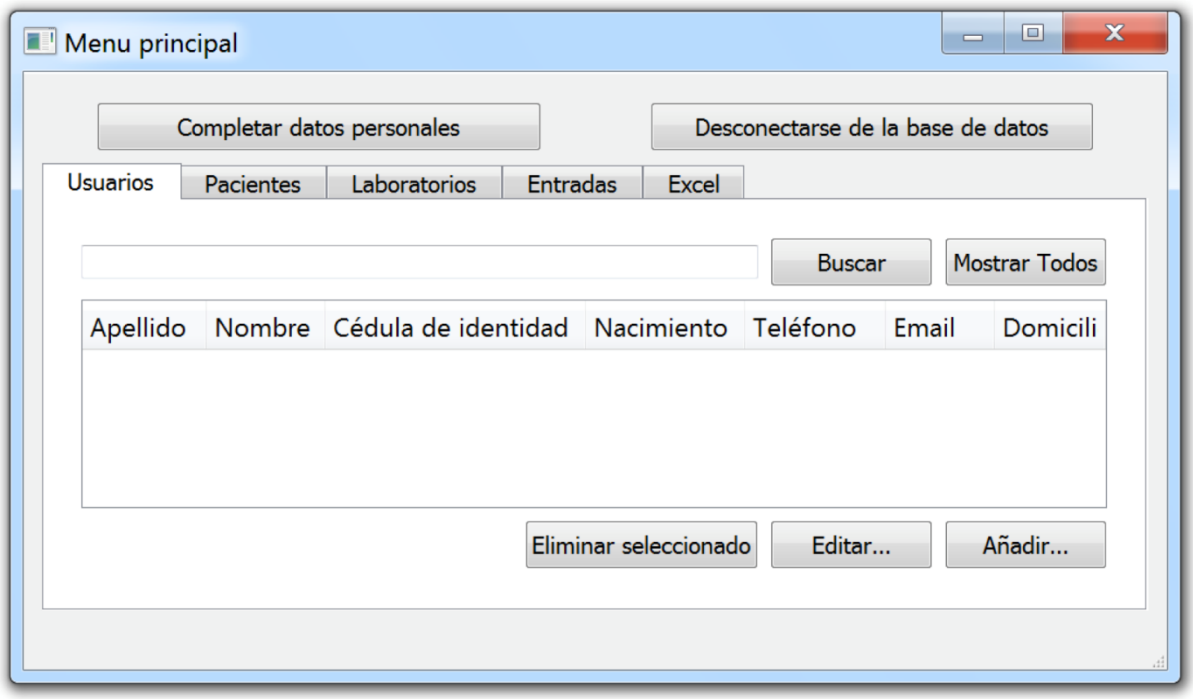


Figure 5

Dans le cas où un de ces deux derniers boutons est enfoncé, une nouvelle fenêtre de dialogue s'ouvre. Cette dernière a le design représenté à la Figure 6.

3.2.1 Fenêtre « Añadir o modificar un usuario»

L'administrateur peut, via cette fenêtre, ajouter un nouvel utilisateur ou encore, octroyer un nouveau mot de passe à un médecin ayant oublié le sien. Il a également la possibilité de changer le laboratoire dans lequel cet usager travaille et de sélectionner ses privilèges en cochant la case ad hoc.

Par défaut, un utilisateur ne peut modifier que les résultats des patientes aux examens effectués au sein du laboratoire auquel il appartient ou uniquement les données gynécologiques dans le cas des centres de santés. Les autres données relatives aux patientes ne peuvent qu'être consultées. Les membres du laboratoire virologique HPV-HIV et des centres de santé de niveau 1, 2 et 3 peuvent également effectuer des modifications au niveau des données personnelles des patientes. Les médecins du laboratoire HP-HIV ont, de plus, l'autorisation de modifier les résultats des tests effectués par le laboratoire d'anatomopathologie. Les privilèges, quant à eux, ne sont accordés à un médecin que si le médecin en question est soit administrateur au sein d'un laboratoire (« Modificar la base de datos » : il a le droit d'ajouter, supprimer ou modifier les caractéristiques du laboratoire et les tests effectués en son sein) soit administrateur général de la base de données (« Gestionar los usuarios » : il est autorisé à ajouter, supprimer ou modifier les utilisateurs et leurs données personnelles). Un utilisateur peut également être et administrateur d'un laboratoire, et administrateur général.

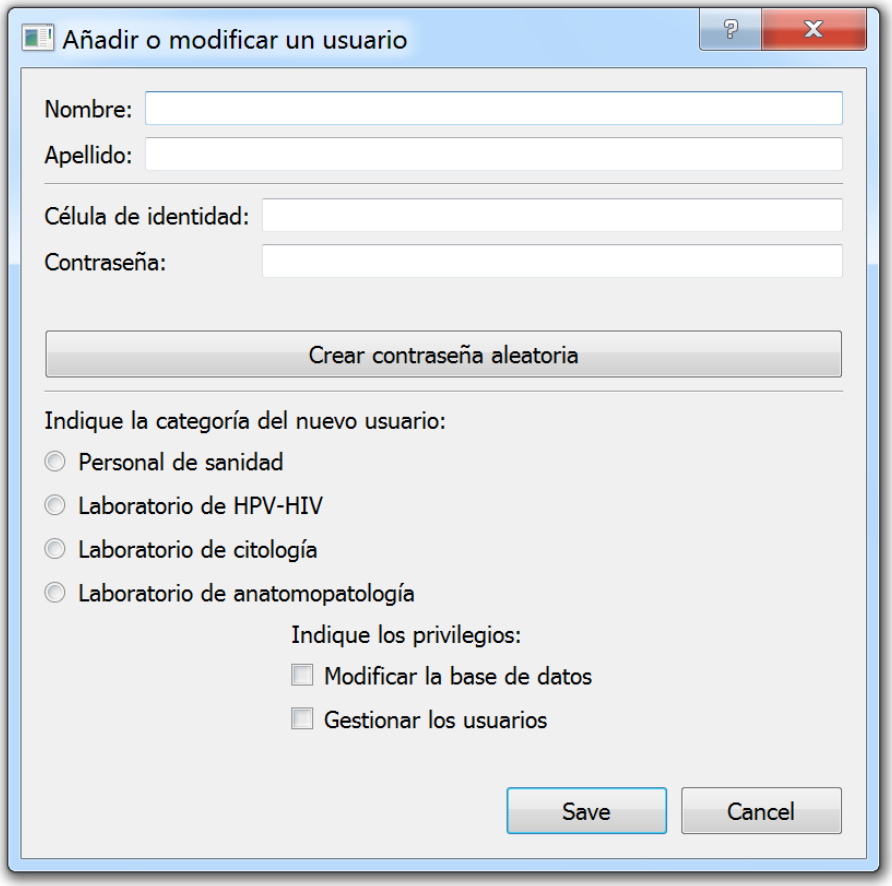


Figure 6

3.3 Onglet « Pacientes »

Cet onglet offre exactement les mêmes possibilités que l'onglet « Usuarios », mais il s'agit dans ce cas d'une liste de patientes. La recherche de patientes au sein de cette liste s'effectue également suivant son nom, prénom ou ID.

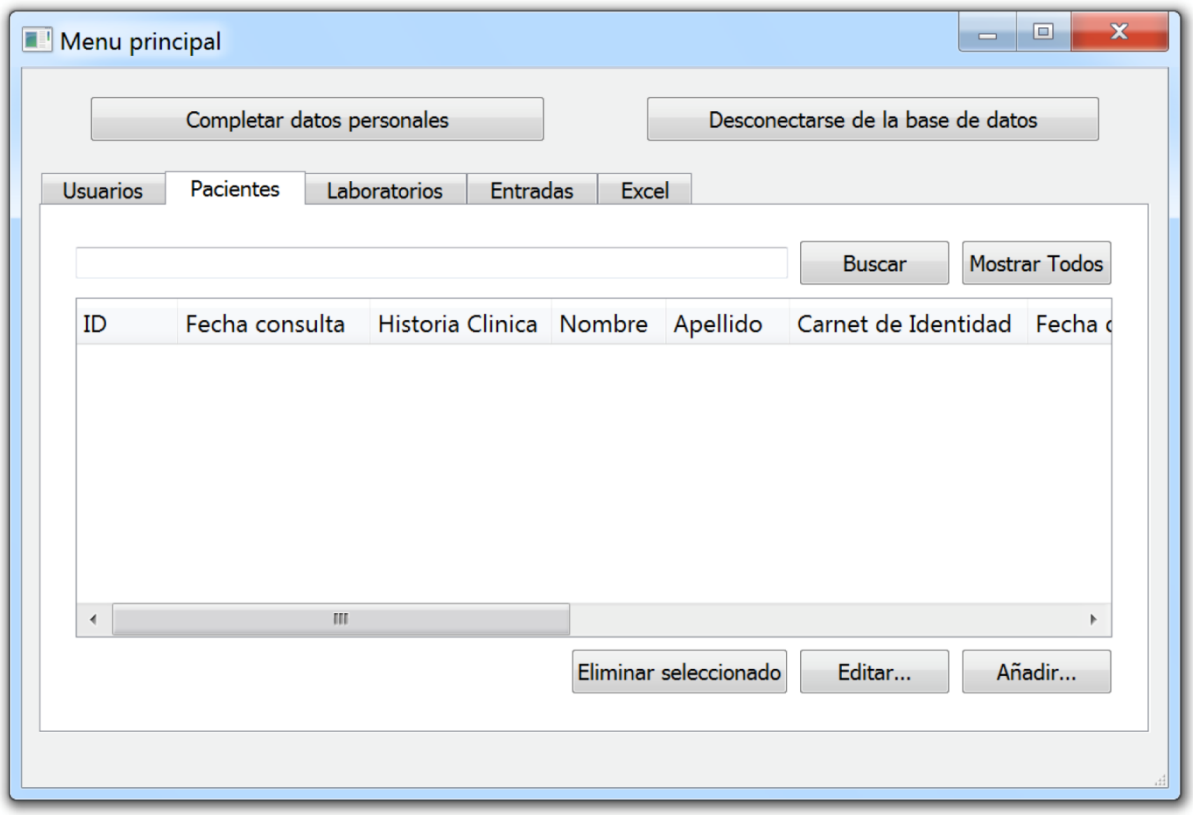


Figure 7

Les actions de suppression, de modification ou d'ajout de patientes sont identiques à celles expliquées à la section précédente mais, pour les deux derniers boutons mentionnés, une fenêtre différente apparaît (cf. Figure 8). Il important de se rendre compte que la suppression d'une patiente au sein de cette liste n'induit pas seulement une suppression de ses données personnelles, mais supprime également tous ses résultats d'examens.

3.3.1 Fenêtre « Añadir o modificar un paciente »

La fenêtre « Añadir o modificar un paciente » permet d'encoder ou visionner les informations personnelles relatives à chaque patiente. L'encodage n'est permis qu'aux membres des centres de santé et du laboratoire HPV-HIV, les autres ne pouvant que consulter les données personnelles. L'utilisateur doit compléter le formulaire en suivant les champs mis en évidence à la Figure 8. Il doit par la suite enregistrer la patiente grâce au bouton « Save ».

Ensuite, il est également possible de compléter des données médicales plus spécifiques à chacun des laboratoires en cliquant sur un des boutons situés au bas de la fenêtre et représentant chacun un laboratoire ou les données gynécologiques. Comme expliqué plus haut, suivant le

laboratoire ou centre de santé auquel appartient l'utilisateur de la base de données, certaines données sont modifiables en plus d'être consultables.

Les résultats des différents tests sont au fur et à mesure ajoutés, complétant ainsi peu à peu le dossier médical de la patiente.

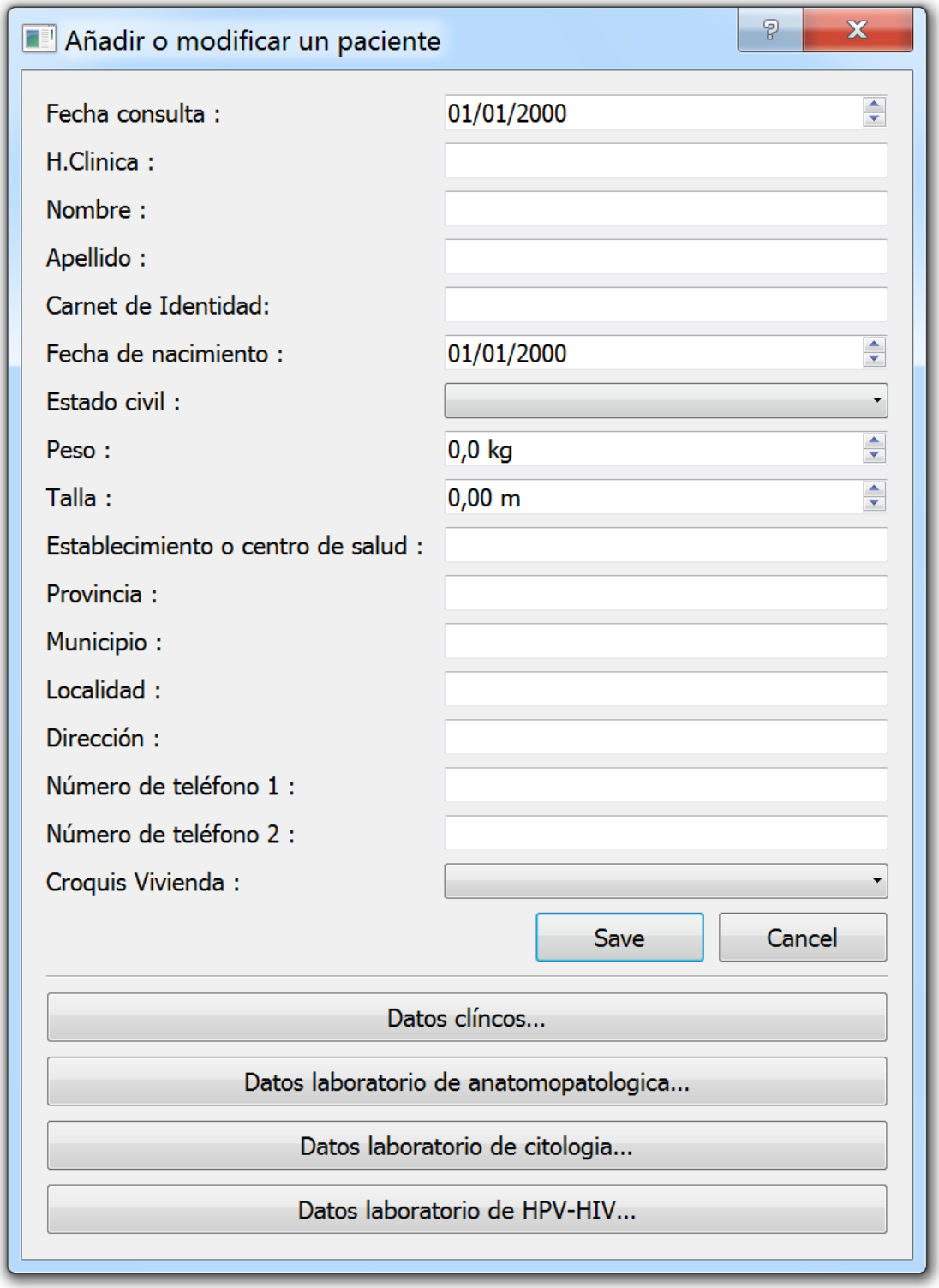


Figure 8

3.3.2 Fenêtre « Añadir o modificar los datos medicos »

Voici les fenêtres (cf. Figures 9 à 12) correspondant aux différents boutons des centres d'examens. Comme précédemment, il est possible de compléter ou de modifier les champs constituant ces interfaces graphiques.

Dans les exemples ci-dessous, les champs sont clairs, ce qui signifie que l'utilisateur a l'autorité nécessaire pour apporter des modifications aux résultats affichés. Cependant, il peut arriver que les champs à modifier soient grisés. Ceci signifie que l'utilisateur n'a pas les privilèges nécessaires pour modifier les données provenant du centre de tests en question. Il peut toutefois en consulter les résultats.

Dans le cas où le clinicien a le droit de modifier les données médicales de la patiente, il ne doit pas oublier de sauvegarder ses modifications ou bien d'annuler en cas d'erreur. Ces deux actions s'effectuent via les deux boutons situés en bas à droite de la fenêtre.

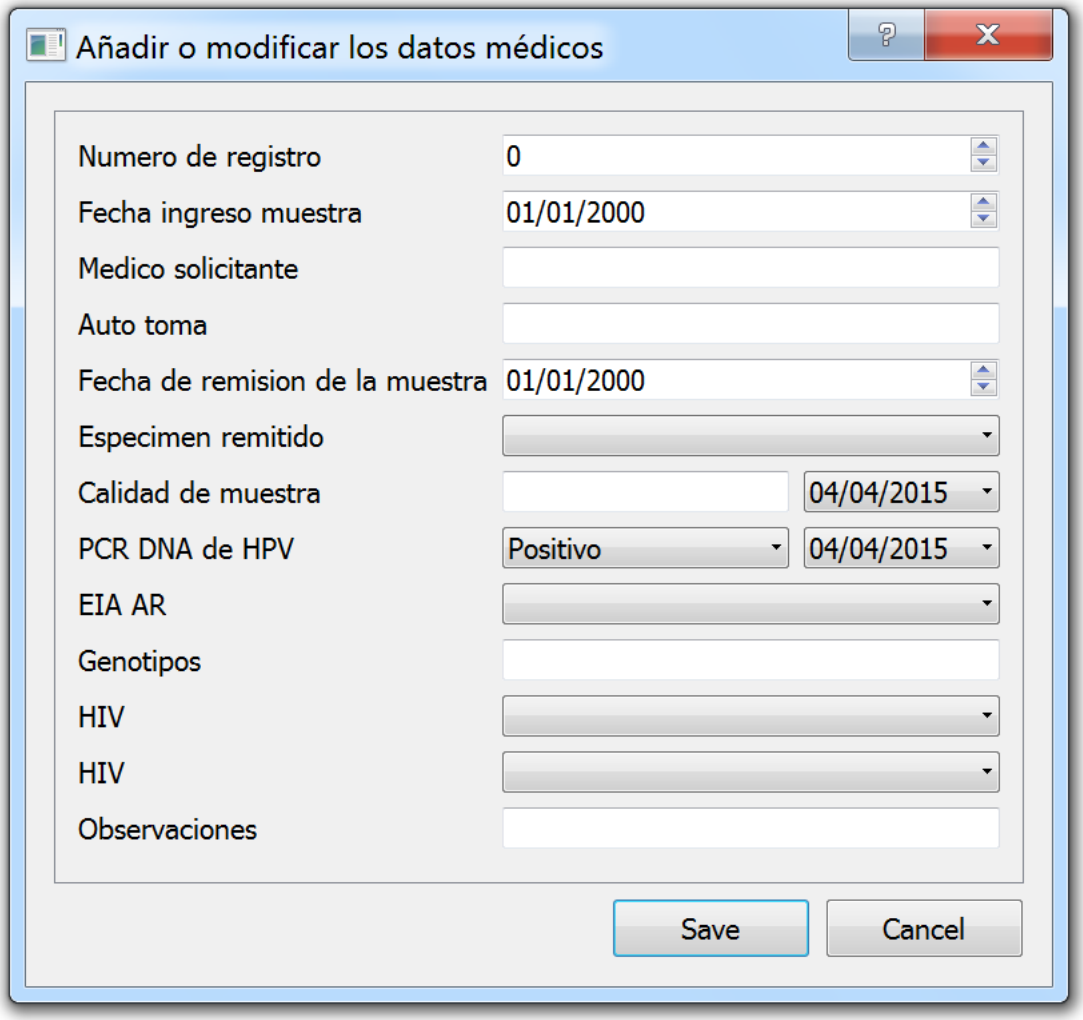


Figure 9

Datos laboratorio de anatomopathologica

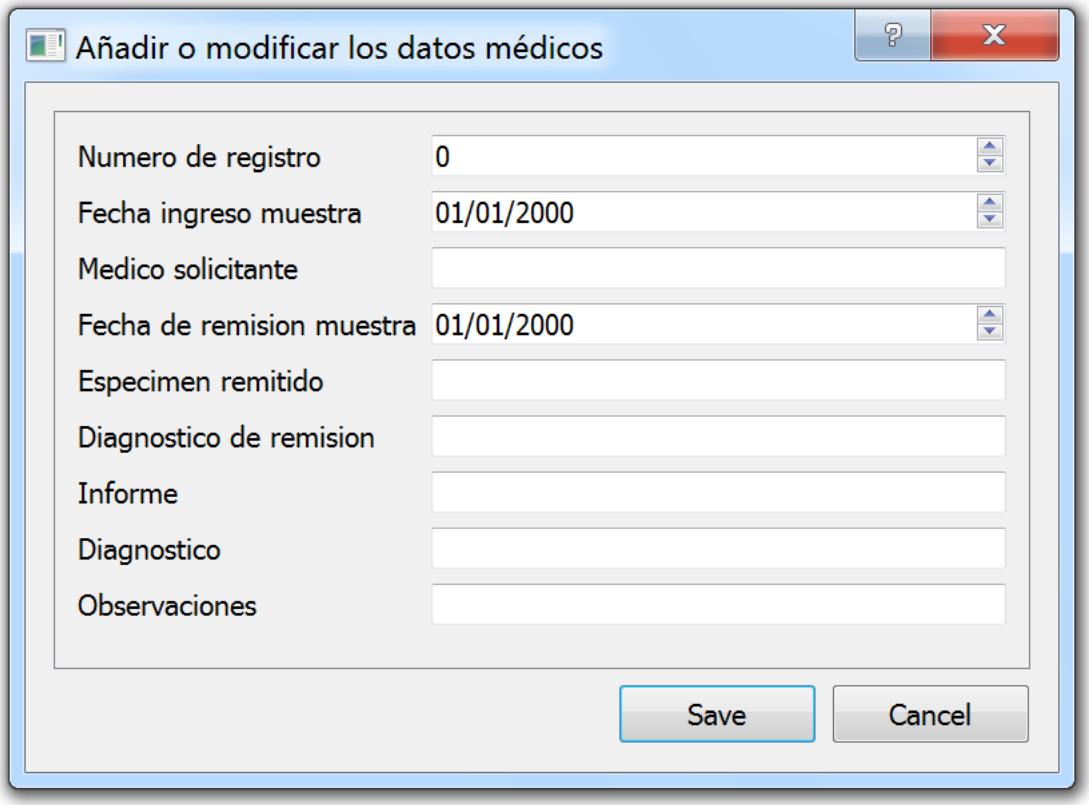


Figure 10

Datos laboratorio de citologia

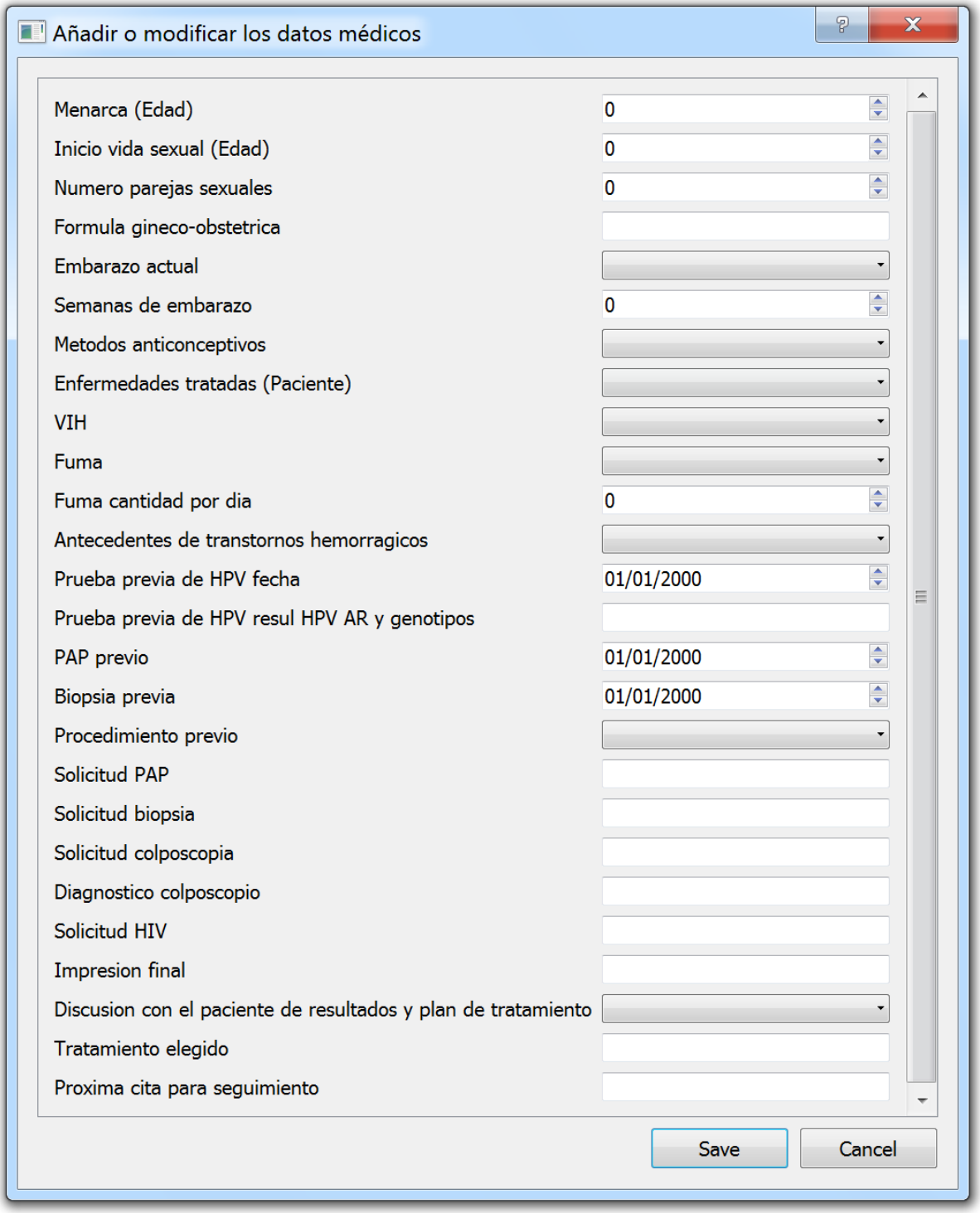


Figure 11

Datos laboratorio de HPV-HIV

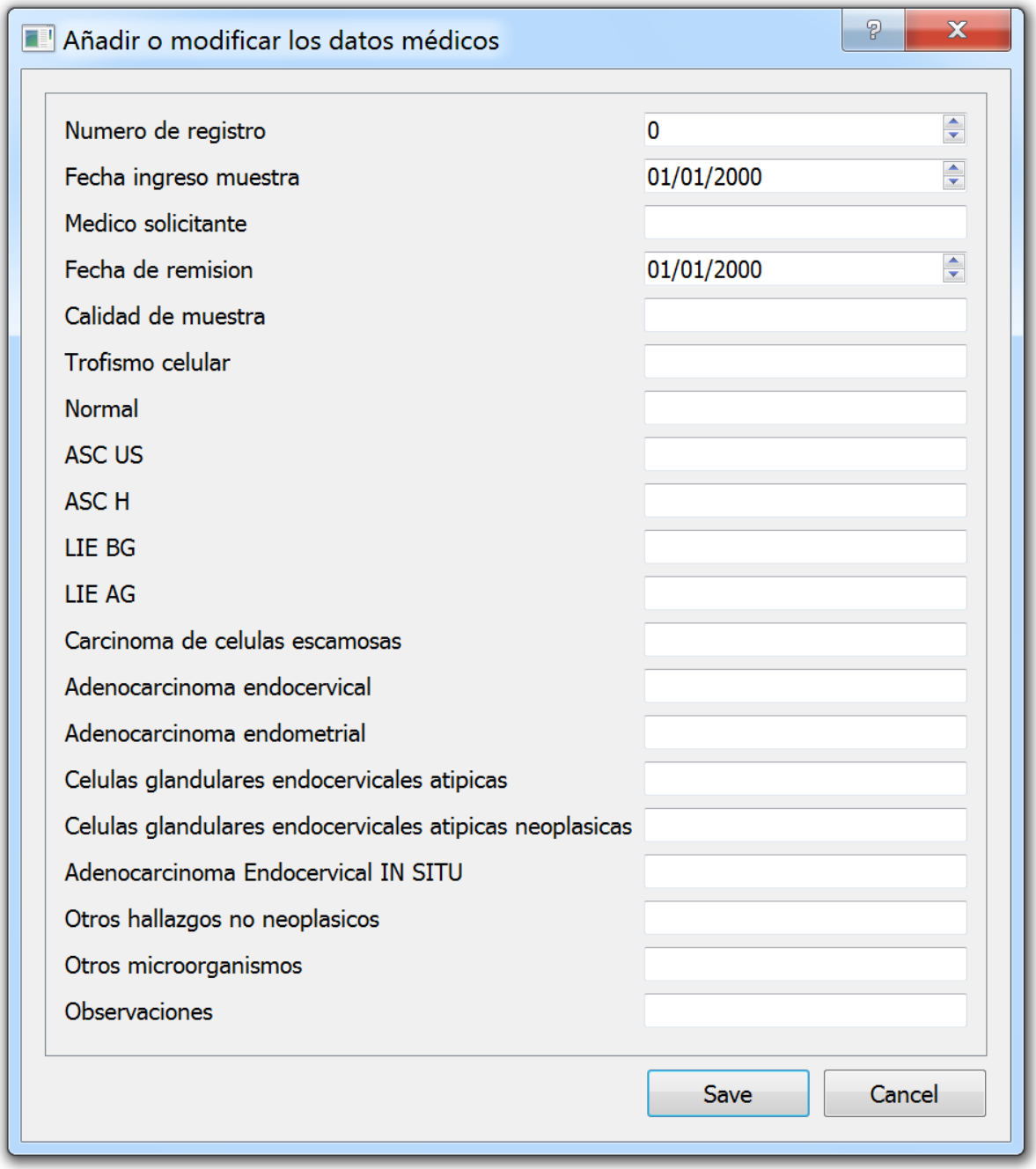


Figure 12

3.4 Onglet « Laboratorios »

Cet onglet (cf. Figure 13) affiche la liste des centres cliniques et des laboratoires et n'est visible que par un administrateur d'un laboratoire ou centre de santé. Les interfaces étant redondantes, encore une fois, les options à sélectionner sont les mêmes que précédemment. Exception faite de la possibilité de supprimer un laboratoire puisque, dans ce cas, un grand nombre de données relatives aux patientes seraient perdues. Si un laboratoire ou hôpital cesse ses activités, les résultats d'examens effectués dans cet établissement sont toujours utiles à conserver pour le suivi et le traitement des patientes. Une autre différence par rapport aux onglets « Usuarios » et « Pacientes » se situe au niveau du moteur de recherche. En effet, ici, la recherche d'un établissement hospitalier ne s'effectue que suivant le nom de ce dernier.

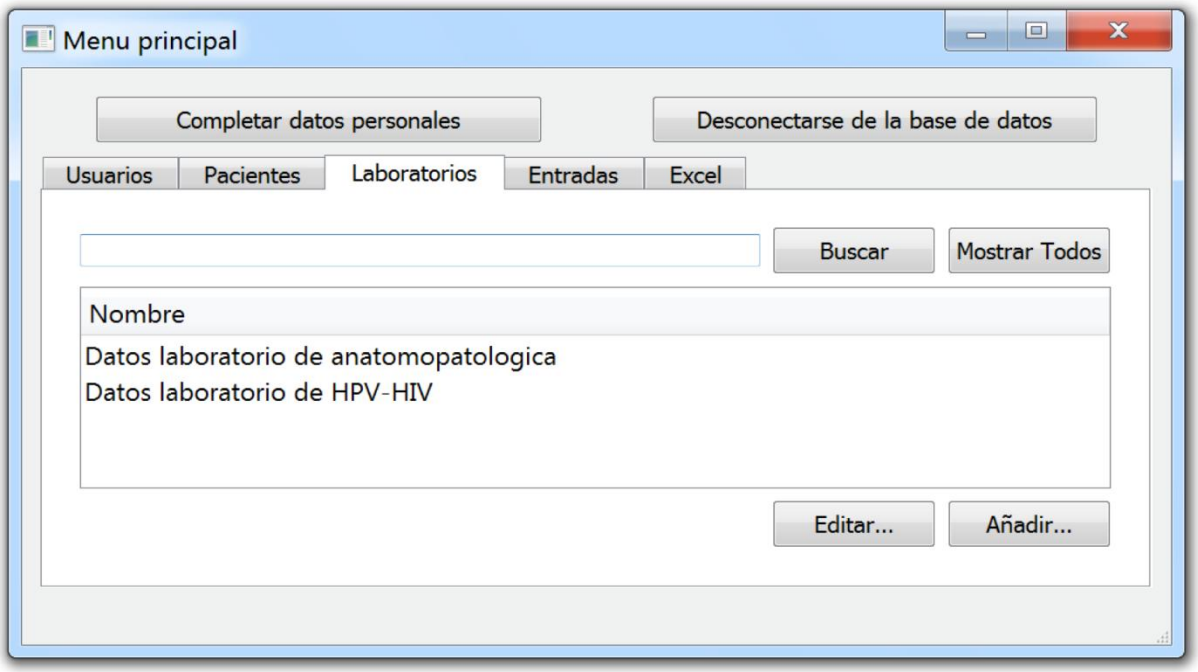


Figure 13

3.4.1 Fenêtre « Añadir o modificar un laboratorio »

Une fois un des boutons « Editar… » ou « Añadir » enfoncé au sein de l'onglet « Laboratorios », la fenêtre suivante s'affiche (cf. Figure 14). Elle permet d'ajouter le nom d'un centre et de l'activer. Cette possibilité d'activer ou non le centre clinique remplace l'option de suppression présente sur les deux précédents onglets. Dans le cas où l'utilisateur décoche le bouton « Activo », un centre est désactivé. Cela signifie que dans cet établissement, les examens qui y sont associés et leurs résultats n'apparaissent plus à l'écran, mais ils résident toujours au sein de la base de données. Il suffit de cocher à nouveau le bouton « Activo » pour voir réapparaître toutes ces données au sein des interfaces graphiques.

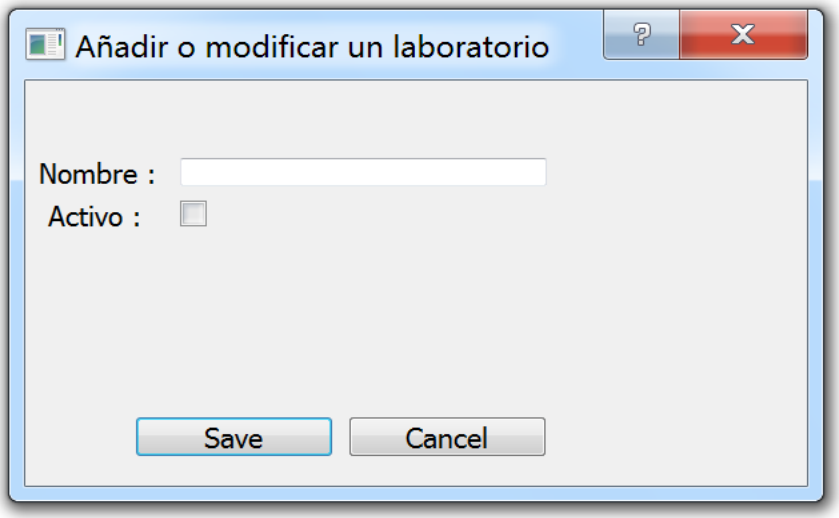


Figure 14

3.5 Onglet « Entradas »

Cet onglet reprend les différents tests provenant des différents laboratoires et est, à nouveau, uniquement visible par un administrateur d'un laboratoire ou centre de santé (Cf. Figure 15). Encore une fois, la liste a été réalisée selon la même structure que les précédentes. Tout comme l'onglet « Laboratorios », celui-ci ne possède pas de bouton supprimer pour éviter de perdre un grand nombre d'informations sur les patientes en cas de suppression d'un examen.

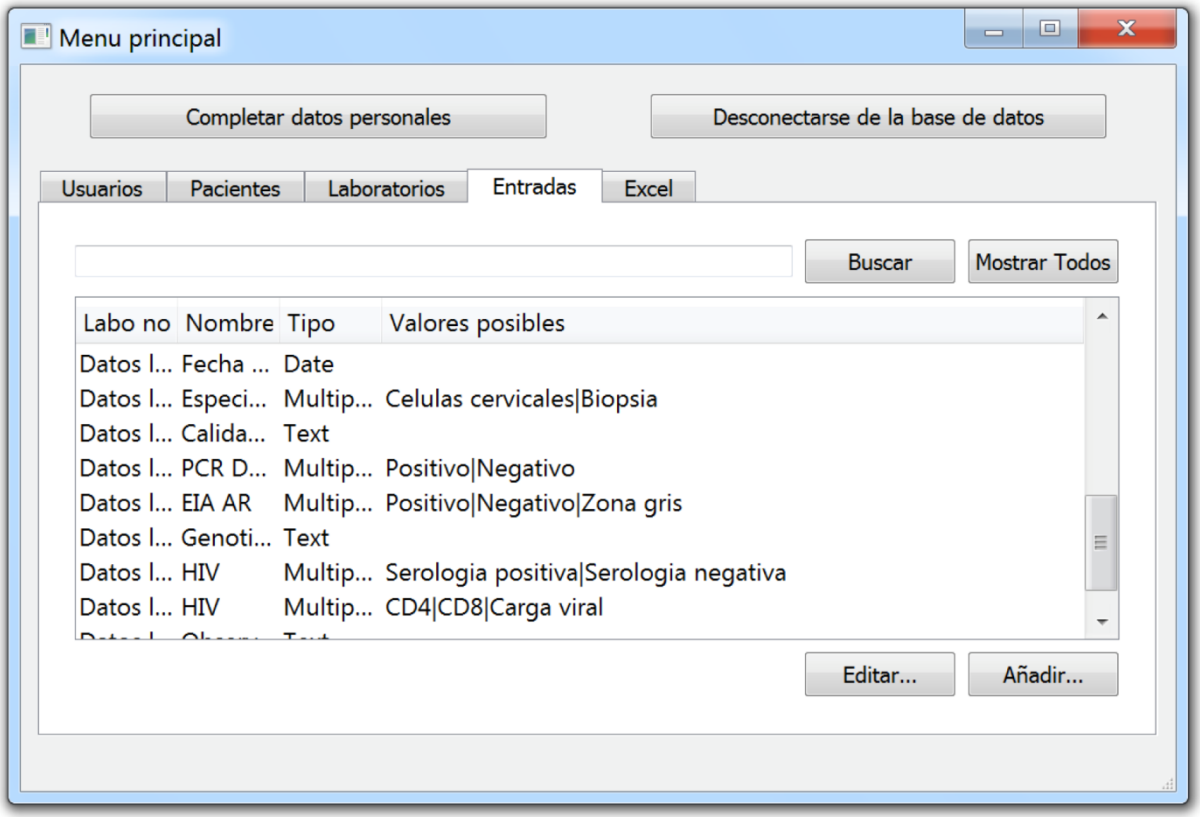


Figure 15

3.5.1 Fenêtre « Añadir o modificar un test »

Dans le cas d'une modification ou d'un ajout de test, la première étape consiste à sélectionner le laboratoire où figurera ce test. Ensuite, il faut choisir un nom adapté au test et sélectionner le type du résultat associé à cet examen (« tipo »). Il est possible de prédéfinir différents types de réponses attendues:

- « Fecha » prédéfinit une interface sous forme de calendrier afin de sélectionner facilement une date ;
- « Float » permet d'encoder un nombre à virgule ;
- « Int » permet d'encoder un nombre entier ;
- « Text » permet d'entrer le texte souhaité ;
- « Multiples resultados » permet d'afficher un menu déroulant avec les différents choix possibles (qu'il ne faut pas oublier de définir dans le champ « Valores posibles »).

Enfin, l'utilisateur peut cocher l'option « Multiples resultados », qui donne la possibilité de pouvoir entrer plusieurs résultats. Un médecin peut également sélectionner le statut du test (actif ou non).

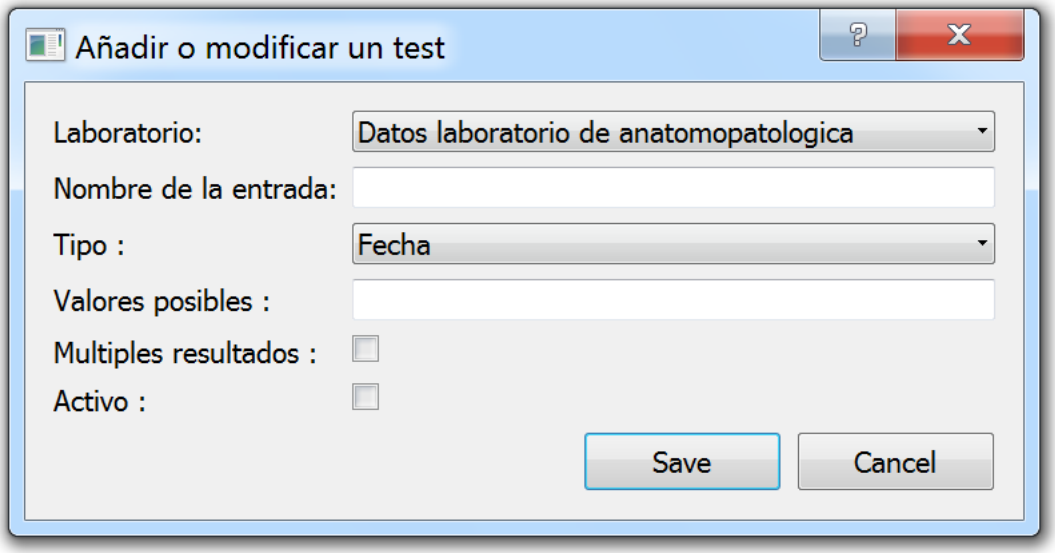


Figure 16

Annexe E : Le manuel utilisateur (version espagnole)

Manual de utilización

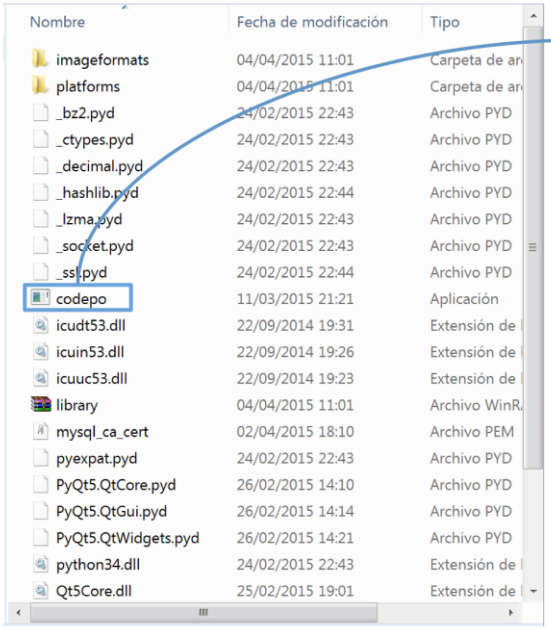
Base de datos para el seguimiento de pacientes susceptibles de padecer cáncer de cuello uterino

ECOLE POLYTECHNIQUE DE BRUXELLES

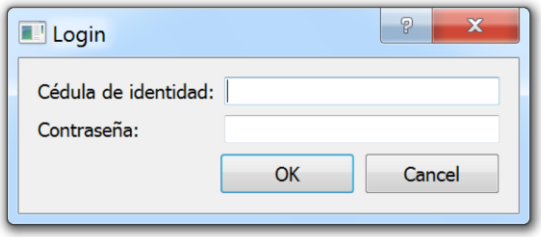
Tabla de contenidos

2

Acceso & Login



Para abrir la aplicación, pulsar dos veces sobre el fichero ejecutable



El acceso a la base de datos requiere identificarse con la Cédula de identidad y la contraseña

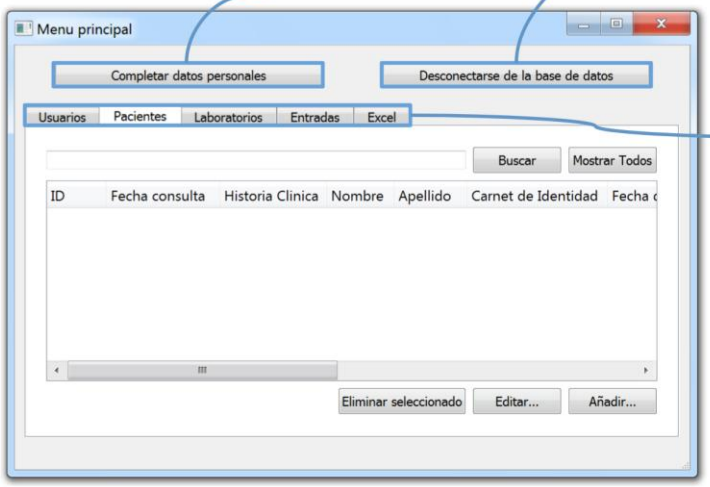
Para obtener una cuenta, se tiene que contactar con un administrador de la base de datos

3

Menu principal

Añadir o modificar los datos personales (5) y cambiar la contraseña de acceso

Cerrar la aplicación una vez se hayan introducido los datos necesarios



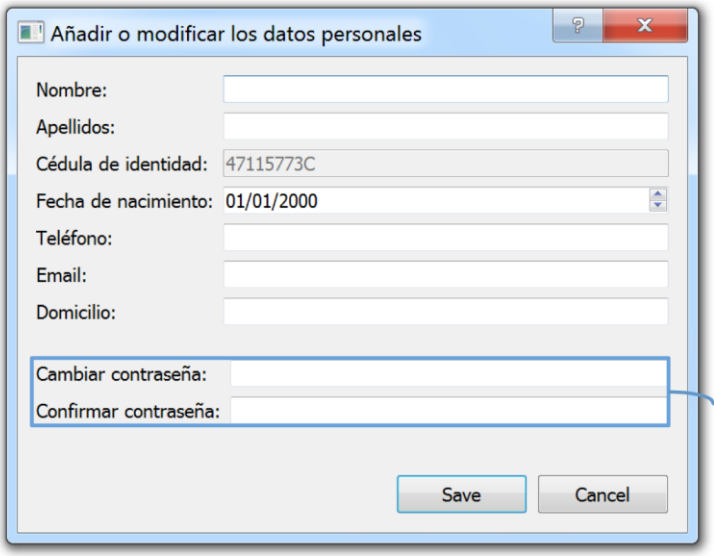
Pulsar en cada una de las distintas pestañas para navegar por los distintos menús

La ventana principal presenta varias pestañas en función de los privilegios otorgados al usuario

Por defecto solo aparecen las pestañas Pacientes y Excel

4

Añadir o modificar los datos personales



Cuando un usuario acceda por primera vez a la base de datos, deberá completar su información personal para facilitar su contacto a los administradores

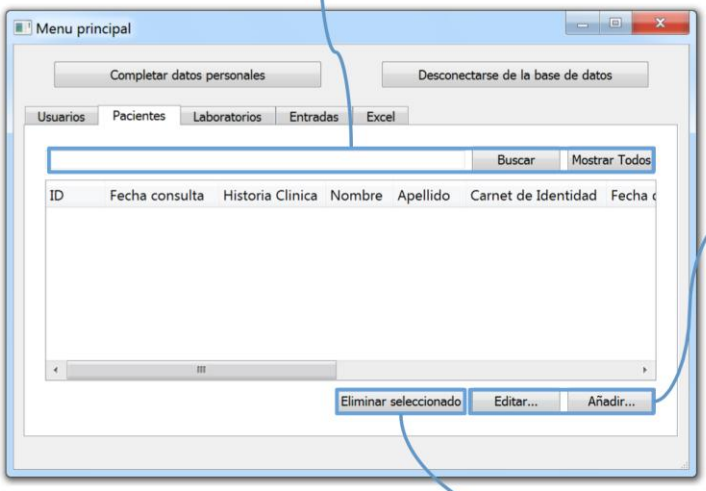
También permite cambiar su contraseña de acceso

Cuando se quiera cambiar la clave acceso, la nueva contraseña tiene que escribirse idénticamente en los dos cuadros de texto. Si los cuadros se dejan vacíos, la clave de acceso no se modificará

5

Menú principal – Pacientes

Buscar pacientes introduciendo su ID, Nombre o Apellido en el cuadro de texto. Para volver a obtener la lista completa, pulse en Mostrar Todos



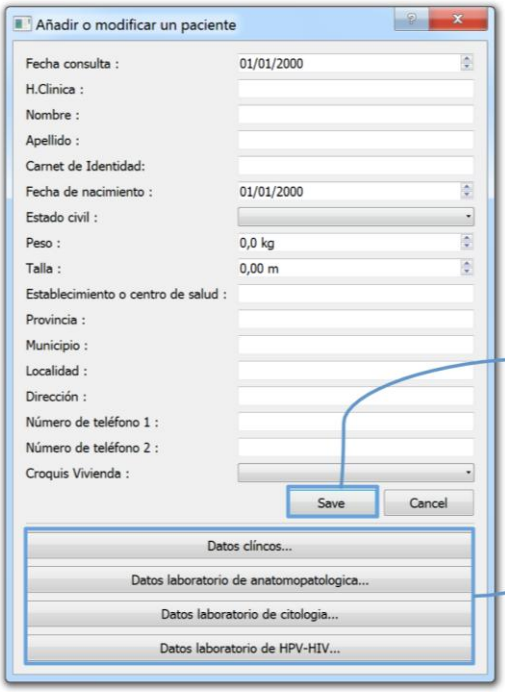
La pestaña Pacientes muestra la lista de los pacientes previamente añadidos y sus datos personales

Una vez seleccionado un paciente, estos botones permiten **Añadir o modificar un paciente (7)**

Un vez eliminado un paciente, toda su información será perdida de forma irrecuperable

6

Añadir o modificar un paciente



La ventana tiene dos partes: La primera son una serie de entradas para los datos personales de un paciente.

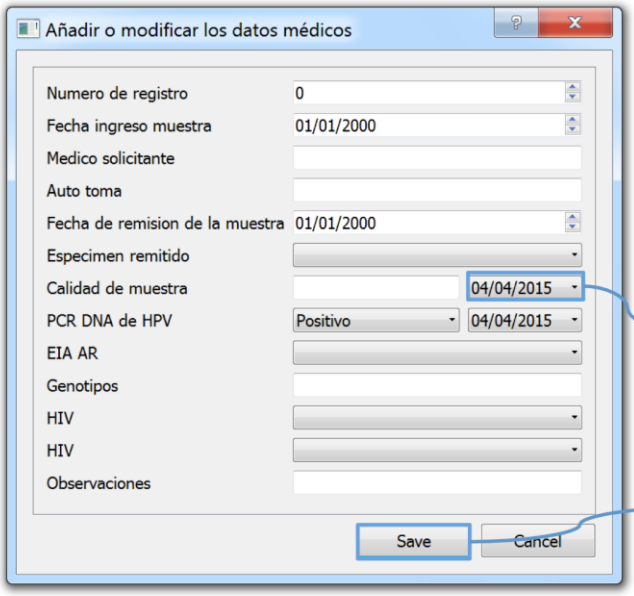
La segunda son una serie de botones, que coinciden con la lista de laboratorios del **Menú principal – Laboratorios (12)**

Guardar los datos añadidos o modificados una vez hayan sido introducidos

Añadir o modificar los datos médicos (8) del laboratorio. Estos botones solo funcionan para los pacientes ya añadidos y guardados en la base de datos

7

Añadir o modificar los datos médicos



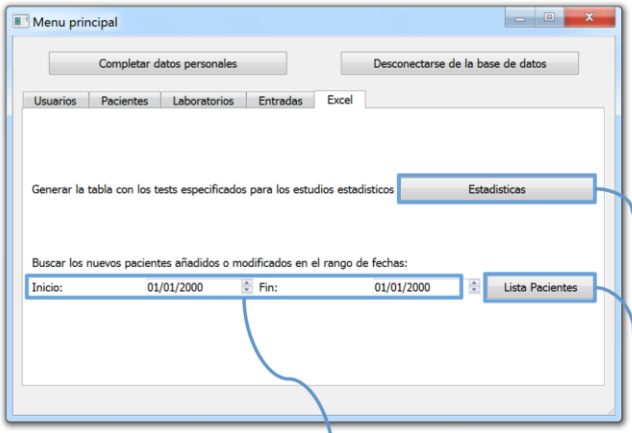
La ventana es un ejemplo de las informaciones y exámenes del laboratorio de HPV-HIV, corresponden a la lista de entradas del **Menú principal – Entradas (14)**

Cuando una entrada tiene un recuadro de fecha al lado, indica que se pueden añadir mas de un resultado para esta entrada. Para verlos, desplegar la lista de fechas y seleccionar la fecha correspondiente al resultado deseado

Guardar los resultados de los exámenes

8

Menú principal – Excel



La pestaña Excel permite generar dos tipos de ficheros CSV, que pueden ser abiertos con Microsoft Excel

Generar una fichero con los datos necesarios para el control estadístico del cáncer de cuello uterino

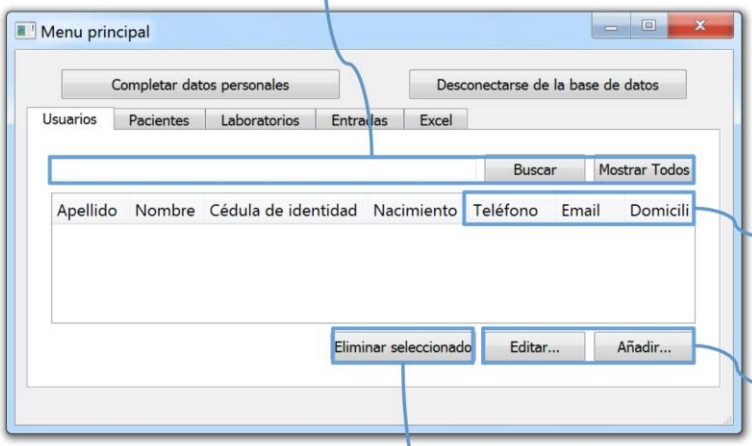
Generar una fichero con los datos que resumen el estado médico de los pacientes añadidos o modificados en el rango de fechas especificado

Indicar el marco de tiempo, especificando el inicio y el fin. En el fichero aparecerán los pacientes añadidos y los pacientes dónde algún dato se ha modificado o añadido dentro del rango de fechas

9

Menú principal – Usuarios

Buscar pacientes introduciendo su Nombre o Apellido en el cuadro de texto. Para volver a obtener la lista completa, pulse en Mostrar Todos



La pestaña Usuarios muestra la lista de los usuarios que tiene acceso a la base de datos y solo aparece a los administradores

Deberá ser actualizada a medida que aparezcan nuevas incorporaciones en los hospitales y laboratorios

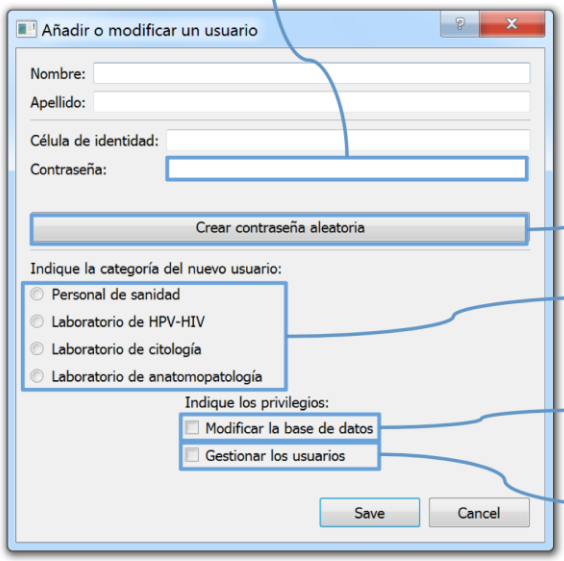
La información complementaría y de contacto de los usuarios deberá ser añadida por ellos mismos cuando tengan acceso por primera vez a la base de datos

Una vez seleccionado un usuario, estos botones permiten **Añadir o modificar un usuario (11)**

Un vez eliminado un usuario, este dejará de tener acceso a la aplicación y toda su información será perdida

10

Añadir o modificar un usuario



Al modificar un usuario, si este campo se deja vacío, su contraseña no cambiará

Crear un numero aleatorio para dar la contraseña de un nuevo usuario, que una vez haya accedido por primera vez, podrá cambiar. El administrador deberá transmitir la contraseña de una forma segura al nuevo usuario

Esta ventana permite añadir un nuevo usuario o modificar su información y privilegios

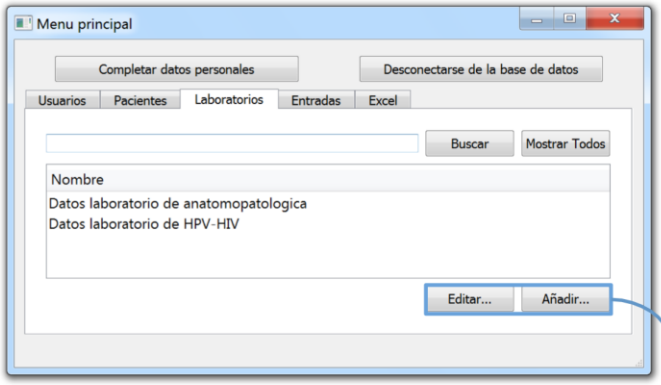
Dependiendo de la categoría escogida, el usuario podrá **Añadir o modificar los datos médicos (8)** de ciertos laboratorios Consultar el **Anexo – Categorías (16)**

Este privilegio da acceso a las pestañas **Laboratorios y entradas – Menú Principal (12 y 14)**

Este privilegio da acceso a las pestaña **Usuarios – Menú Principal (10)** y por tanto convierte al usuario en administrador

11

Menú principal – Laboratorios



La pestaña Laboratorios solo aparece a los usuarios con el privilegio de modificar la base de datos en sí misma

Los laboratorios aparecen como botones en la ventana **Añadir o modificar un paciente (7)**

Su uso requiere una particular cuidado ya que los laboratorios añadidos, para evitar errores en la base de datos, no pueden ser eliminados, solo desactivados.

Una vez seleccionado un usuario, estos botones permiten **Añadir o modificar un laboratorio (13)**

12

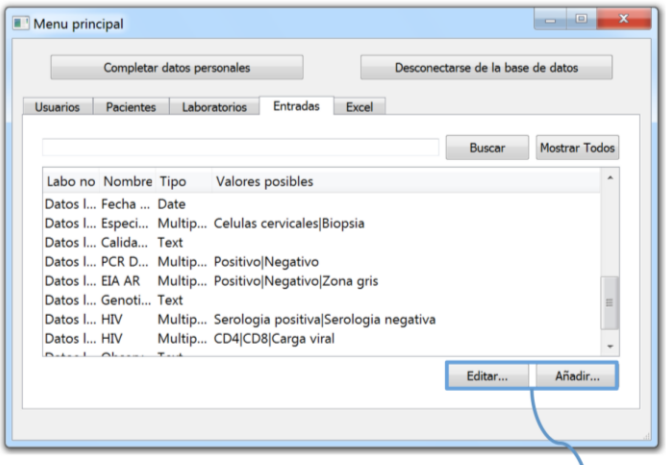
Añadir o modificar un laboratorio

Esta ventana permite añadir o modificar un laboratorio, determinando su nombre y si este debe aparecer en la base de datos o no

Si el parámetro Activo esta marcado, este aparecerá como un botón en la ventana **Añadir o modificar un paciente (7)**

13

Menú principal – Entradas



La pestaña Entradas solo aparece a los usuarios con el privilegio de modificar la base de datos en sí misma

Las entradas aparecen en la ventana **Añadir o modificar los datos médicos (8)**

Su uso requiere una particular cuidado ya que las entradas añadidas, para evitar errores en la base de datos, no pueden ser eliminadas, solo desactivadas

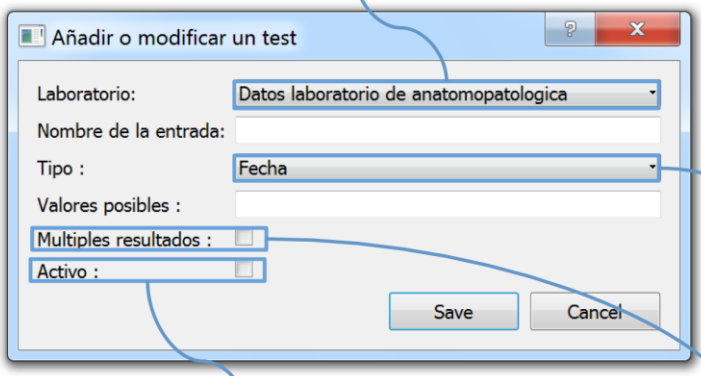
Una vez seleccionado una entrada, estos botones permiten **Añadir o modificar una entrada (15)**

14

Añadir o modificar una entrada

Especificar en la lista desplegable el laboratorio al que va a pertenecer la entrada

Esta ventana permite editar una entrada de la base de datos y establecer sus características



Especificar en la lista el formato del resultado de la entrada

Escoger si una entrada debe guardar distintos resultados. Al lado de la entrada aparecerá una lista con fechas, y se podrá asociar a cada fecha un resultado.

Marcando esta casilla, la entrada aparecerá en la ventana **Añadir o modificar datos médicos (8)**

15

Anexo – Categorías

Todos los usuarios pueden ver la información personal y médica de los pacientes. Pero, dependiendo de su categoría, un usuario puede modificar o no ciertos datos, como esta especificado en la tabla

Una columna marcada indica que el usuario de la categoría puede modificar los datos de la columna

Categoría	Datos personales	Datos clíncos	Anatomo-patología	Citología	HPV-HIV
Personal de sanidad	✓	✓			
Laboratorio de anatomopatología			✓		
Laboratorio de citología				✓	
Laboratorio HPV-HIV	✓		✓		✓

Si se añade un nuevo laboratorio, todos los usuarios, independientemente de su categoría, podrán modificar los datos que contenga

16